

# Banking Industry Architecture Network

## BIAN Metamodel Specification

## Copyright

© Copyright 2013 by BIAN Association. All rights reserved.

THIS DOCUMENT IS PROVIDED "AS IS," AND THE ASSOCIATION AND ITS MEMBERS, MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THIS DOCUMENT ARE SUITABLE FOR ANY PURPOSE; OR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. NEITHER THE ASSOCIATION NOR ITS MEMBERS WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THIS DOCUMENT UNLESS SUCH DAMAGES ARE CAUSED BY WILFUL MISCONDUCT OR GROSS NEGLIGENCE. THE FOREGOING DISCLAIMER AND LIMITATION ON LIABILITY DO NOT APPLY TO, INVALIDATE, OR LIMIT REPRESENTATIONS AND WARRANTIES MADE BY THE MEMBERS TO THE ASSOCIATION AND OTHER MEMBERS IN CERTAIN WRITTEN POLICIES OF THE ASSOCIATION.

# TABLE OF CONTENTS

Introduction .....	1
Purpose .....	1
Scope .....	1
Overview .....	1
Updates from Previous Version .....	1
Package – BIAN .....	2
1. Cross-Level Elements .....	2
Class BIANElement .....	2
<i>Direct Superclasses</i> .....	3
<i>Associations</i> .....	3
<i>Attributes</i> .....	3
<i>Constraints</i> .....	3
Class RelatedExternalElement .....	3
<i>Direct Superclasses</i> .....	3
<i>Associations</i> .....	3
<i>Attributes</i> .....	3
<i>Constraints</i> .....	4
A. Service Landscape Viewpoint .....	4
Class BIANBusinessArea .....	5
<i>Direct Superclasses</i> .....	6
<i>Associations</i> .....	6
<i>Attributes</i> .....	6
<i>Constraints</i> .....	6
Class BIANConstraint .....	6
<i>Direct Superclasses</i> .....	6
<i>Associations</i> .....	7
<i>Attributes</i> .....	7
<i>Constraints</i> .....	7
Class BusinessDomain .....	7
<i>Direct Superclasses</i> .....	7
<i>Associations</i> .....	7
<i>Attributes</i> .....	7
<i>Constraints</i> .....	7
Class Postcondition .....	8
<i>Direct Superclasses</i> .....	8
<i>Associations</i> .....	8
<i>Attributes</i> .....	8
<i>Constraints</i> .....	8
Class Precondition .....	8
<i>Direct Superclasses</i> .....	8
<i>Associations</i> .....	8
<i>Attributes</i> .....	9
<i>Constraints</i> .....	9
Class ServiceDomain .....	9
<i>Direct Superclasses</i> .....	9
<i>Associations</i> .....	9
<i>Attributes</i> .....	10
<i>Constraints</i> .....	10
Class ServiceGroup .....	10
<i>Direct Superclasses</i> .....	10
<i>Associations</i> .....	10
<i>Attributes</i> .....	11
<i>Constraints</i> .....	11

Class ServiceOperation.....	11
<i>Direct Superclasses</i> .....	11
<i>Associations</i> .....	11
<i>Attributes</i> .....	12
<i>Constraints</i> .....	13
Class ServiceRole .....	13
<i>Direct Superclasses</i> .....	14
<i>Associations</i> .....	14
<i>Attributes</i> .....	14
<i>Constraints</i> .....	14
Enumeration InvocationKind .....	14
<i>Literals</i> .....	14
<b>B1. Capabilities-Responsibilities-Delegation Viewpoint.....</b>	<b>15</b>
Class Capability.....	15
<i>Direct Superclasses</i> .....	15
<i>Associations</i> .....	15
<i>Attributes</i> .....	16
<i>Constraints</i> .....	16
Class Delegation .....	16
<i>Direct Superclasses</i> .....	16
<i>Associations</i> .....	16
<i>Attributes</i> .....	16
<i>Constraints</i> .....	16
Class Notification.....	17
<i>Direct Superclasses</i> .....	17
<i>Associations</i> .....	17
<i>Attributes</i> .....	17
<i>Constraints</i> .....	17
Class RequestSatisfaction .....	17
<i>Direct Superclasses</i> .....	17
<i>Associations</i> .....	18
<i>Attributes</i> .....	18
<i>Constraints</i> .....	18
Class Responsibility .....	18
<i>Direct Superclasses</i> .....	18
<i>Associations</i> .....	18
<i>Attributes</i> .....	18
<i>Constraints</i> .....	18
Class ResponsibilityFulfillment.....	18
<i>Direct Superclasses</i> .....	19
<i>Associations</i> .....	19
<i>Attributes</i> .....	19
<i>Constraints</i> .....	19
<b>B2. Capabilities-Responsibilities-Delegation with Level 2 .....</b>	<b>20</b>
<b>C. Business Object Viewpoint.....</b>	<b>21</b>
Class BIANBusinessAssociationEnd .....	21
<i>Direct Superclasses</i> .....	22
<i>Associations</i> .....	22
<i>Attributes</i> .....	22
<i>Constraints</i> .....	22
Class BIANBusinessAttribute .....	22
<i>Direct Superclasses</i> .....	22
<i>Associations</i> .....	22
<i>Attributes</i> .....	22
<i>Constraints</i> .....	22
Class BusinessObject .....	23

<i>Direct Superclasses</i> .....	23
<i>Associations</i> .....	23
<i>Attributes</i> .....	24
<i>Constraints</i> .....	24
<b>D. Message Viewpoint</b> .....	<b>24</b>
Class BIANChoiceComponent .....	25
<i>Direct Superclasses</i> .....	25
<i>Associations</i> .....	25
<i>Attributes</i> .....	25
<i>Constraints</i> .....	25
Class BIANMessageAssociationEnd .....	25
<i>Direct Superclasses</i> .....	25
<i>Associations</i> .....	26
<i>Attributes</i> .....	26
<i>Constraints</i> .....	26
Class BIANMessageAttribute .....	26
<i>Direct Superclasses</i> .....	26
<i>Associations</i> .....	26
<i>Attributes</i> .....	26
<i>Constraints</i> .....	26
Class BIANMessageBuildingBlock .....	26
<i>Direct Superclasses</i> .....	27
<i>Associations</i> .....	27
<i>Attributes</i> .....	27
<i>Constraints</i> .....	27
Class BIANMessageComponent .....	27
<i>Direct Superclasses</i> .....	27
<i>Associations</i> .....	27
<i>Attributes</i> .....	27
<i>Constraints</i> .....	27
Class BIANMessageDefinition .....	27
<i>Direct Superclasses</i> .....	28
<i>Associations</i> .....	28
<i>Attributes</i> .....	28
<i>Constraints</i> .....	28
<b>E. Scenario Viewpoint</b> .....	<b>29</b>
Class BIANMessageTransmission .....	30
<i>Direct Superclasses</i> .....	31
<i>Associations</i> .....	31
<i>Attributes</i> .....	31
<i>Constraints</i> .....	32
Class BusinessScenario .....	35
<i>Direct Superclasses</i> .....	35
<i>Associations</i> .....	35
<i>Attributes</i> .....	36
<i>Constraints</i> .....	36
Enumeration MessageTransmissionKind .....	36
<i>Literals</i> .....	36
<b>F. Implementation Component Viewpoint</b> .....	<b>37</b>
Class ImplementationComponent .....	37
<i>Direct Superclasses</i> .....	37
<i>Associations</i> .....	37
<i>Attributes</i> .....	38
<i>Constraints</i> .....	38
Class ImplementationInterface .....	38
<i>Direct Superclasses</i> .....	38

<i>Associations</i> .....	38
<i>Attributes</i> .....	38
<i>Constraints</i> .....	38
G. Canonical Instances vs. Non-Canonical Instances .....	39
Package – BIAN::Level1 .....	39
Additional Classes .....	40
Enumerations .....	40
Package – BIAN::Level2 .....	40
Additional Classes .....	40
Enumerations .....	40
Package – BIAN::Level3 .....	40
Additional Classes .....	40
Package – BIAN_ISO20022 .....	41
A. BIAN-ISO20022 Service Landscape Viewpoint.....	42
Class BusinessRole .....	42
Class Constraint .....	42
Class RepositoryConcept.....	42
B. BIAN-ISO20022 Capabilities-Responsibilities-Delegation Viewpoint .....	43
Class BusinessComponent .....	43
C. BIAN-ISO20022 Business Object Viewpoint .....	44
Class BusinessAssociationEnd .....	44
Class BusinessAttribute .....	44
D. BIAN-ISO20022 Message Viewpoint .....	45
Class ChoiceComponent.....	45
Class MessageAssociationEnd .....	45
Class MessageAttribute .....	46
Class MessageBuildingBlock .....	46
Class MessageComponent .....	46
Class MessageDefinition.....	46
E. BIAN-ISO20022 Scenario Viewpoint.....	46
Class BusinessProcess .....	47
Class BusinessProcessTrace .....	47
Class BusinessRoleTrace .....	47
Class BusinessTransaction .....	47
Class MessageTransmission .....	47
Class MessageTypeTrace .....	47
Class Participant .....	47
Class Receive .....	48
Class Send .....	48
Package – BIAN_SemanticMetadata .....	48
BIAN - Semantic Metadata Binding .....	49
Class TaggableElement .....	49
<i>Direct Superclasses</i> .....	49
<i>Associations</i> .....	49
<i>Attributes</i> .....	50
<i>Constraints</i> .....	50
Package – ISO20022_SemanticMetadata .....	50
Binding to RepositoryConcept .....	51
Class SemanticMetadataTag .....	51
<i>Direct Superclasses</i> .....	51
<i>Associations</i> .....	51
<i>Attributes</i> .....	51
<i>Constraints</i> .....	52

Class SemanticRole .....	52
<i>Direct Superclasses</i> .....	52
<i>Associations</i> .....	52
<i>Attributes</i> .....	52
<i>Constraints</i> .....	53
Package – SemanticMetadata .....	53
Semantic Metadata.....	53
Class BusinessContextType .....	54
<i>Direct Superclasses</i> .....	54
<i>Associations</i> .....	54
<i>Attributes</i> .....	54
<i>Constraints</i> .....	54
Class EmbeddedTag.....	54
<i>Direct Superclasses</i> .....	54
<i>Associations</i> .....	54
<i>Attributes</i> .....	55
<i>Constraints</i> .....	55
Class SemanticGrammarRole.....	55
<i>Direct Superclasses</i> .....	55
<i>Associations</i> .....	55
<i>Attributes</i> .....	55
<i>Constraints</i> .....	55
Class StandaloneTag .....	56
<i>Direct Superclasses</i> .....	56
<i>Associations</i> .....	56
<i>Attributes</i> .....	56
<i>Constraints</i> .....	56
SemanticMetadata::Examples .....	56
A. Example - Tags Embedded in the Definition of a Service Operation.....	57
B. Example - Additional Semantic Role: Property.....	58
C. Example - Additional Semantic Role: Object Class Qualifier .....	59
D. Example - Tags based on Compound Vocabulary Entries.....	60
E. Example - Illustration that Semantic Roles are Defined Only Once .....	61
F. Example - Business Context Tags .....	62
G. Example - Referencing Externally Defined Business Context Tags.....	63
H. Example - Functional Pattern Tags.....	64
Appendix A: Tree.....	65
Class Hierarchy .....	65
Enumeration Hierarchy.....	70

## TABLE OF FIGURES

Figure 1.	1. Cross-Level Elements.....	2
Figure 2.	A. Service Landscape Viewpoint .....	4
Figure 3.	B1. Capabilities-Responsibilities-Delegation Viewpoint.....	15
Figure 4.	B2. Capabilities-Responsibilities-Delegation with Level 2.....	20
Figure 5.	C. Business Object Viewpoint.....	21
Figure 6.	D. Message Viewpoint.....	24
Figure 7.	E. Scenario Viewpoint.....	29
Figure 8.	F. Implementation Component Viewpoint .....	37
Figure 9.	G. Canonical Instances vs. Non-Canonical Instances .....	39
Figure 10.	A. BIAN-ISO20022 Service Landscape Viewpoint.....	42
Figure 11.	B. BIAN-ISO20022 Capabilities-Responsibilities-Delegation Viewpoint .....	43
Figure 12.	C. BIAN-ISO20022 Business Object Viewpoint .....	44
Figure 13.	D. BIAN-ISO20022 Message Viewpoint .....	45
Figure 14.	E. BIAN-ISO20022 Scenario Viewpoint.....	46
Figure 15.	BIAN - Semantic Metadata Binding .....	49
Figure 16.	Binding to RepositoryConcept .....	51
Figure 17.	Semantic Metadata.....	53
Figure 18.	A. Example - Tags Embedded in the Definition of a Service Operation.....	57
Figure 19.	B. Example - Additional Semantic Role: Property.....	58
Figure 20.	C. Example - Additional Semantic Role: Object Class Qualifier .....	59
Figure 21.	D. Example - Tags based on Compound Vocabulary Entries.....	60
Figure 22.	E. Example - Illustration that Semantic Roles are Defined Only Once.....	61
Figure 23.	F. Example - Business Context Tags.....	62
Figure 24.	G. Example - Referencing Externally Defined Business Context Tags.....	63
Figure 25.	H. Example - Functional Pattern Tags.....	64



# Introduction

---

## Purpose

This document is a specification of the BIAN Metamodel Version 2.0. Its intended audience is software architects who maintain the BIAN architecture and developers of tools and templates that are based on the metamodel. The intended audience does not include business analysts who define and use BIAN service definitions, who should be shielded from the metamodel's technical detail by templates and tools.

However, the definitions of the basic concepts represented in the metamodel -- such as definitions of Business Area, Business Domain, Service Domain, Service Group, Responsibility, Capability, Delegation, etc. -- could, with some rewording, be extracted to become entries in the vocabulary of business concepts that BIAN will be creating going forward.

## Scope

Although the document contains some explanatory material, it is a reference document, not a user guide. BIAN also publishes a "How-To Guide."

Although the BIAN Metamodel is an extension of the ISO20022 Metamodel, this document does not contain the specification of the ISO 20022 Metamodel, which is available from ISO.

## Overview

This document is organized according to the hierarchical package structure of the metamodel. The order in which the packages are presented is based on the alphabetical order of the package names.

At the beginning of the section corresponding to a package, the UML diagrams contained in the package are displayed and explained. Then the classes that appear in the diagrams are defined, except that if a class was already covered previously in the document because it appeared in a diagram that lives in a previously documented package, the definition of the class is not repeated.

## Updates from Previous Version

There are no major changes from Version 1.6 to 2.0, only minor updates and bug fixes.

## Package – BIAN

The BIAN package contains the elements of the core BIAN Metamodel and the diagrams that provide visual projections of those elements. This package excludes metamodel elements and diagrams that support business vocabularies, which are contained in the SemanticMetadata package and the BIAN\_SemanticMetadata package. It also excludes the BIAN\_ISO20022 package, which has to do with the binding of the BIAN Metamodel to the ISO20022 Metamodel.

### Viewpoint Diagrams

Each viewpoint diagram below is a projection of a subset of the metamodel's elements that are relevant to a particular point of view. The viewpoints are not entirely disjoint. For example, ServiceDomain appears in more than one viewpoint because of its centrality in the overall service landscape.

## 1. Cross-Level Elements

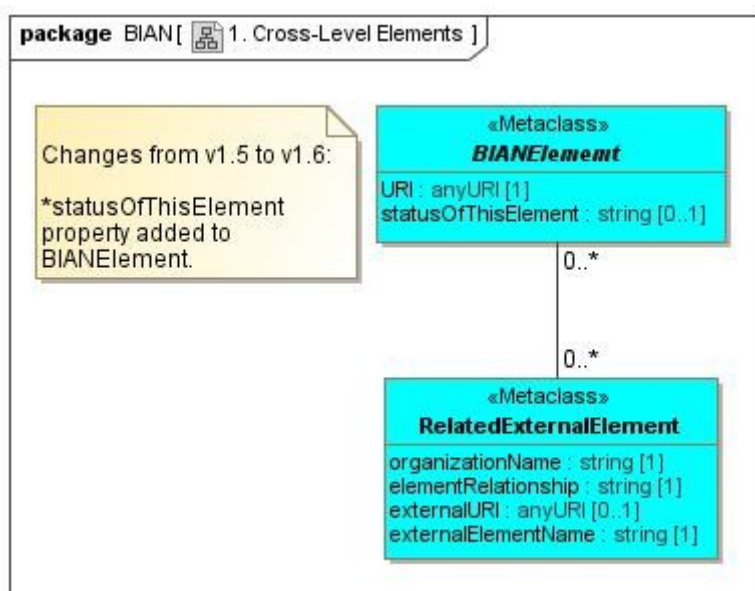


Figure 1. 1. Cross-Level Elements

This diagram shows the elements of the BIAN Metamodel that are not specific to any of the three BIAN architectural levels.

All elements of the BIAN Metamodel inherit from the abstract class **BIANElement**; thus all BIAN elements have URIs and can be associated with elements defined in external repositories such as the ISO 20022 Repository.

### Class **BIANElement**


```
package BIAN
```

An element in a BIAN service definition.

This abstract class is at the top of the BIAN Metamodel's inheritance hierarchy. Thus, all BIAN elements have URIs and can be associated with elements defined in external repositories such as the ISO 20022 Repository.

### Direct Superclasses

### Associations

 `relatedExternalElement : RelatedExternalElement [0..*]`

Elements defined outside of BIAN that are relevant to the BIANElement.

### Attributes

`URI : anyURI [1]`

The BIANElement's URI.

`statusOfThisElement : string [0..1]`

Free text describing the status of the element, beyond what is already specified by the `registrationStatus` property inherited from ISO 20022.

### Constraints

## Class RelatedExternalElement

`package BIAN`

An element that is defined outside of BIAN but is relevant to a BIANElement. For example, a `BIANMessageDefinition` in a `BIAN ServiceOperation` may be related to an element in the ISO20022 Repository.

### Direct Superclasses

### Associations

 `bianElement : BIANElement [0..*]`

The BIANElements to which the external element is related.

### Attributes

`organizationName : string [1]`

The name of the external organization that manages the external element.

`elementRelationship : string [1]`

A description of the relationship between the BIANElement and the external element.

For example, the relationship could be that the BIANElement has been submitted to the ISO20022 Repository, or that the BIANElement reuses an element of the ISO20022 Repository.

externalURI : anyURI [0..1]

The external element's URI.

externalElementName : string [1]

The name of the external element.

### Constraints

## A. Service Landscape Viewpoint

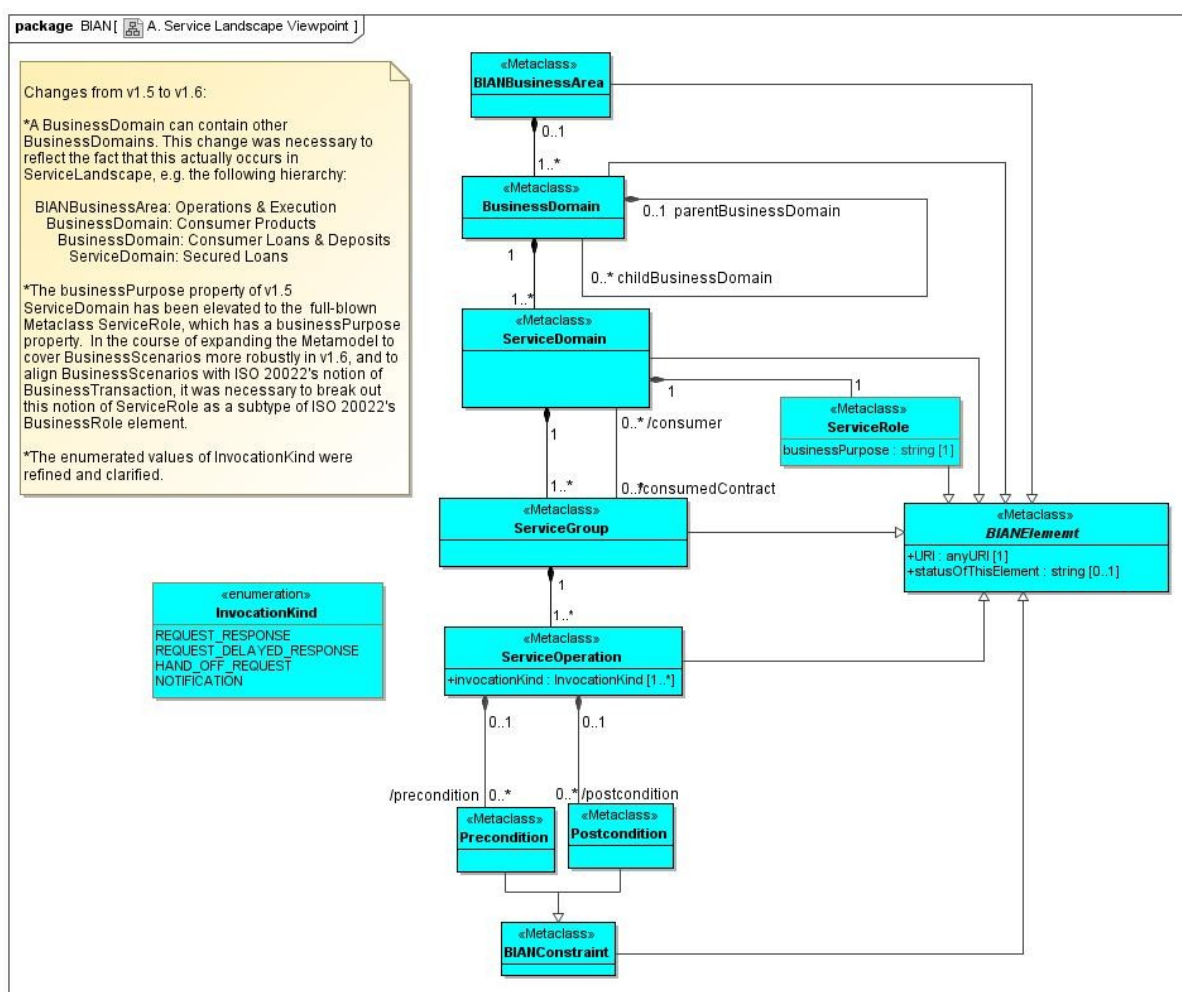


Figure 2. A. Service Landscape Viewpoint

This diagram highlights key elements of the service landscape.

The ultimate goal of BIAN is to arrive at a set of standardized IT services for banking. A first approach was to get agreement on a common application landscape that could be used as a framework to position the standardized Service Operations. However, we quickly realized that the existing application landscapes of the BIAN banks were too different to allow consolidation into a common application landscape. Indeed, these application landscapes were developed over a long period of time in a rather isolated way. They started out looking like the business world, but gradually diverged

from it for multiple reasons: available technology, people and budgets, strategic decisions, mergers and acquisitions, opportunities, and many other accidental events.

To reap the benefits of the agility and interoperability promises of SOA, implementation independence is a key aspect. BIAN therefore headed towards getting a consensus on a common logical functional landscape, consisting of coherent sets of logical capabilities/functionalities. It is called the Service Landscape (and not IT function landscape) to emphasize its role as a framework for supporting the definition and management of the BIAN Service Operations.

By progressively refining this Service Landscape, we have a systematic top-down approach to obtaining elementary functions that expose capabilities through Service Operations. It should be noted that, in reality, the approach BIAN follows is a combination of top-down and bottom-up: we heavily draw upon experiences from the BIAN participants for identifying candidate Service Operations and use the top-down approach mainly for checking consistency, completeness, detecting missing services, etc.

In fact, the Service Landscape is a very powerful instrument for many reasons:

- it acts as an “index” to the service catalogue, offering overview and facilities for discovery and look up
- it helps in organizing the management of Service Operations, assigning ownership
- it is the basic instrument for service portfolio management, i.e. managing the complete set of services as a whole, allowing reporting, monitoring, impact analysis, trend surveys, usage statistics, completeness and consistency checks, etc.
- it serves as a reference framework for migration roadmaps (by projecting existing and target application landscape on the service landscape) or for gap analysis (by projecting a packaged solution and the existing application landscape on the service landscape)

In view of the key role it plays in the BIAN activities and for the management of Service Operations in general, we consider the Service Landscape to be a major deliverable of BIAN. [Currently no comparable standard – besides individual or commercial models - is known within BIAN]

Within BIAN, we identified the need for distinguishing three levels in the IT function tree, each of them having a specific role:

- Business Area
- Business Domain
- Service Domain

BIAN does not define specific Business Areas and Business Domains as canonical standards. It is not possible to force a particular hierarchical decomposition of business function on all banks. The BusinessAreas and Business Domains that BIAN publishes in its service landscape comprise a reference model only. The BIAN Service Domains, on the other hand, are canonical, and are designed to fit into an arbitrary number of Business Area - Business Domain hierarchies.

### Class BIANBusinessArea

```
package BIAN::Level1
```

A BIANBusinessArea is formed by a broad set of capabilities and responsibilities and lies at the highest level of the service landscape hierarchy. BIANBusinessAreas are used to decompose the functions of financial institutions. This decomposition is primarily driven by business understanding and complemented by application and information-specific needs.

#### Comments

- "BIAN" was added to the name of this class in order to make its distinction from ISO20022 MessageComponent obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages. Note that, unlike other cases where "BIAN" has been added to the name of an element of the

BIAN Metamodel to distinguish the element from an ISO 20022 element, in this case BIANBusinessArea is **not** a subtype of ISO 20022's BusinessArea element; ISO 20022 BusinessAreas are normatively defined by ISO and have a certain structure defined by the ISO 20022 Metamodel, whereas BIANBusinessAreas are non-normatively defined by BIAN as a vehicle -- along with BusinessDomains -- for organizing the BIAN Service Domains; conformance with BIAN does not require adopting BIAN's specific hierarchy of BIANBusinessAreas and BusinessDomains, said hierarchy being merely a reference model.

- 
- BIAN has identified (amongst others) the following BIANBusinessAreas as part of BIAN's reference model of business areas and business domains:
  - Reference Data
  - Sales & Service
  - Operations & Execution
  - Analytics
  - Business Support
- BusinessAreas are decomposed (i.e. subdivided) into Business Domains.
- BusinessAreas are part of the Service Landscape of BIAN.

### Direct Superclasses

[BIANElement](#), [RepositoryConcept](#), [TaggableElement](#)

### Associations



businessDomain : [BusinessDomain](#) [1..\*]

The BusinessDomains that fall within the BIANBusinessArea. The association is a composition in which BIANBusinessArea plays the role of composite and BusinessDomain plays the role of component.

### Attributes

### Constraints

## Class BIANConstraint

```
package BIAN::Level1
```

BIANConstraint is a specialization of ISO20022 Constraint.

What makes BIANBusinessAttribute different from ISO20022 BusinessAttribute is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties from those two superclasses.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 Constraint obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

### Direct Superclasses

[BIANElement](#), [Constraint](#)

### Associations

### Attributes

### Constraints

## Class BusinessDomain

package BIAN::Level1

A BusinessDomain is an element of the functional decomposition of the banking business functions in the context of the Service Landscape. Business Domains are linked to certain skills and knowledge, which are clearly identifiable in the banking business.

### Comments

- A BusinessDomain belongs to exactly one BIANBusinessArea
- BusinessDomains are part of the Service Landscape

### Direct Superclasses

[BIANElement](#), [RepositoryConcept](#), [TaggableElement](#)


### Associations

 businessArea : [BIANBusinessArea](#) [0..1]


The BIANBusinessArea to which the BusinessDomain belongs. The association is a composition in which BIANBusinessArea plays the role of composite and BusinessDomain plays the role of component.

 serviceDomain : [ServiceDomain](#) [1..\*]

The ServiceDomains that fall within the BusinessDomain. The association is a shared aggregation in which BusinessDomain plays the role of aggregate and ServiceDomain plays the role of aggregatee.

 childBusinessDomain : [ServiceDomain](#) [1..\*]

The ServiceDomains that fall within the BusinessDomain. The association is a shared aggregation in which BusinessDomain plays the role of aggregate and ServiceDomain plays the role of aggregatee.

 parentBusinessDomain : [ServiceDomain](#) [1..\*]

The ServiceDomains that fall within the BusinessDomain. The association is a shared aggregation in which BusinessDomain plays the role of aggregate and ServiceDomain plays the role of aggregatee.

### Attributes

### Constraints

Constraint ParentIsBusinessAreaOrBusinessDomain:

The parent of a BusinessDomain must be either a BusinessArea or another BusinessDomain.

```
[parentBusinessDomain.oclIsUndefined() xor businessArea.oclIsUndefined()]
```

### Class Postcondition

```
package BIAN::Level2
```

A Postcondition specifies a condition of the system that must be true at the time of completion of some reference behavior. Reference behaviors include the execution of a ServiceOperation or the behavior triggered by a BIANMessageTransmission. A Postcondition is distinct from a Precondition, which must be true when the behavior starts.

Postcondition is a subclass of BIANConstraint.

#### *Direct Superclasses*

[BIANConstraint](#), [TaggableElement](#)

#### *Associations*



serviceOperationConstrainedbyPostcondition : [ServiceOperation](#) [0..1]

The ServiceOperation that the Postcondition constrains. The association is a composition in which ServiceOperation plays the role of composite and Postcondition plays the role of component.



: [BIANMessageTransmission](#) [0..1]

#### *Attributes*

#### *Constraints*

### Class Precondition

```
package BIAN::Level2
```

A Precondition specifies a condition of the system that must be true at the start of some reference behavior. Reference behaviors include the execution of a ServiceOperation or the behavior triggered by a BIANMessageTransmission. A Precondition is distinct from a Postcondition, which must be true when the behavior completes.

Precondition is a subclass of BIANConstraint.

#### *Direct Superclasses*

[BIANConstraint](#), [TaggableElement](#)

#### *Associations*



✓ serviceOperationConstrainedByPrecondition : [ServiceOperation](#) [0..1]

The ServiceOperation that the Precondition constrains. The association is a composition in which ServiceOperation plays the role of composite and Precondition plays the role of component.

✓ transmissionConstrainedByPrecondition : [BIANMessageTransmission](#) [0..1]

### Attributes

### Constraints

## Class ServiceDomain

package BIAN::Level1

A ServiceDomain represents an 'atomic' logical design. Atomic means that a BIAN Service Domain represents the smallest practical capability or functional partition that can be service-enabled as a discrete and unique business capability.

All BIAN Service Domains taken together make up a 'peer set' with each performing its own specific business function or purpose. BIAN arranges this collection of Service Domains into a reference framework called the BIAN Service Landscape.

A ServiceDomain owns a set of ServiceGroups, each of which owns a set of ServiceOperations. A ServiceDomain also owns a BusinessObject called its focusObject.

### Direct Superclasses

[BIANElement](#), [RepositoryConcept](#), [TaggableElement](#)

### Associations

✓ businessDomain : [BusinessDomain](#) [1]

The BusinessDomain within which the ServiceDomain is grouped.

✓ providedContract : [ServiceGroup](#) [1..\*]

The ServiceGroups that the ServiceDomain is contractually obligated to provide. The association is a composition in which ServiceDomain plays the role of composite and ServiceGroup plays the role of component.

✓ consumedContract : [ServiceGroup](#) [0..\*]

The ServiceGroups that other ServiceDomains provide and that this ServiceDomain consumes.

✓ responsibility : [Responsibility](#) [1..\*]

The Responsibilities that the ServiceDomain assumes. The association is a composition in which ServiceDomain plays the role of composite and Responsibility plays the role of component.

✍ focusObject : [BusinessObject](#) [1]

The BusinessObject whose instances are managed by the ServiceDomain.

✍ referencedObject : [BusinessObject](#) [0..\*]

The BusinessObjects whose instances the ServiceDomain references but does not manage.

✍ delegation : [Delegation](#) [0..\*]

The Delegations that target this ServiceDomain.

✍ notification : [Notification](#) [0..\*]

The Notifications to which this ServiceGroup subscribes.

☑ : [ServiceRole](#) [1]

### Attributes

### Constraints

## Class ServiceGroup

package BIAN::Level2

A ServiceGroup is a set of ServiceOperations, and is owned by a ServiceDomain. In essence, it is an interface to the ServiceDomain that is defined in terms of business semantics rather than in technical IT terms.

### Direct Superclasses

[BIANElement](#), [RepositoryConcept](#), [TaggableElement](#)

### Associations

☑ provider : [ServiceDomain](#) [1]

The ServiceDomain that is contractually obligated to provide the ServiceGroup's ServiceOperations. The association is a composition in which ServiceDomain plays the role of composite and ServiceGroup plays the role of component.

☑ serviceOperation : [ServiceOperation](#) [1..\*]

The ServiceOperations that make up the ServiceGroup.

✍ consumer : [ServiceDomain](#) [0..\*]

The ServiceDomains that consume the ServiceGroup's ServiceOperations when delegating a Responsibility.

✍ realization : [ImplementationInterface](#) [0..\*]

The ImplementationInterfaces that realize the ServiceGroup.

### Attributes

### Constraints

## Class ServiceOperation

package BIAN::Level2

A ServiceOperation represents a service defined at the level of business semantics, specifying the access to one or more capabilities of a ServiceDomain.

### Comments

- ServiceOperations are grouped into ServiceGroups.
- A ServiceOperation has a provider and consumers.
- Information is passed in and out of a ServiceOperation by the means of BIANMessageDefinitions.

### Related Concepts

- WS\*-Standards [W3C] define service operations via WSDL
- OASIS SOA Reference Model [OASIS]

### Direct Superclasses

[BIANElement](#), [RepositoryConcept](#), [TaggableElement](#)

### Associations

✍ serviceGroup : [ServiceGroup](#) [1]

The ServiceGroup to which the Operation belongs. The association is a composition in which ServiceGroup plays the role of composite and ServiceOperation plays the role of component.

✍ inputMessage : [BIANMessageDefinition](#) [0..1]

The BIANMessageDefinition that serves as the ServiceOperation's input parameter.

✍ outputMessage : [BIANMessageDefinition](#) [0..1]

The BIANMessageDefinition that serves as the ServiceOperation's output parameter.

✍ faultMessage : [BIANMessageDefinition](#) [1]

The BIANMessageDefinition that serves as the ServiceOperation's fault BIANMessageDefinition.

✓ realizedDelegation : [Delegation](#) [0..\*]

The Delegations that ServiceDomains realize by consuming this ServiceOperation.

✓ realizedCapability : [Capability](#) [1..\*]

The Capabilities that are carried out via this ServiceOperation.

✓ precondition : [Precondition](#) [0..\*]

The Preconditions that constrain this ServiceOperation.

A Precondition of a ServiceOperation specifies a condition of the system that must be true when the ServiceOperation is invoked by a service consumer. It is distinct from a Postcondition, which must be true after a ServiceOperation has been executed.

Precondition is a subclass of BIANConstraint.

### Comments

- As we describe ServiceOperations at the level of business semantics, the Preconditions are of a business nature.
- The Preconditions of ServiceOperations are typically expressed in terms of the BusinessObjects that are involved in the execution of the ServiceOperation.
- The consumer of a service typically is responsible for ensuring that Preconditions are met, whereas the service provider ensures / guarantees an outcome described via the Postconditions.

✓ postcondition : [Postcondition](#) [0..\*]

The Postconditions that constrain this ServiceOperation.

A Postcondition of a ServiceOperation specifies a condition of the system that must be true when the ServiceOperation has completed executing. It is distinct from a Precondition, which is a condition that must be true when the ServiceOperation is called by a service consumer.

Postcondition is a subclass of BIANConstraint.

### Comments

- As we describe ServiceOperations at the level of business semantics, the Postconditions are of a business nature. They are distinct from technical postconditions, which have to be specified in a technical specification of a ServiceOperation.
- Postconditions of ServiceOperations are typically expressed in terms of the BusinessObjects that are involved in the execution of the ServiceOperation.
- Whereas the consumer of a service is responsible for ensuring that Preconditions are met, the service provider ensures / guarantees an outcome described via the Postconditions.

### Attributes

invocationKind : [InvocationKind](#) [1..\*]

The ServiceOperation's invocation style. There are four specific invocation styles, represented by the four allowable values specified by the [InvocationKind](#) enumeration.

### Constraints

Constraint Delegation-ServiceDomainAlignment:

The target ServiceDomain of a Delegation that the ServiceOperation realizes is the ServiceDomain that owns the ServiceOperation.

```
[realizedDelegation.target = self.serviceGroup.provider  
]
```

Constraint Capability-ServiceDomainAlignment:

Each Capability that the ServiceOperation realizes fulfills a Responsibility of the same ServiceDomain that owns the ServiceOperation

```
[realizedCapability->forAll  
(  
    fulfilledResponsibility.responsibleServiceDomain =  
    self.serviceGroup.provider  
)  
]
```

Constraint PreconditionDerivation:

The precondition property is derived by using the constraint property inherited from ISO20022's RepositoryConcept element. The derivation rule is: start with the set of all Constraints that pertain to the ServiceOperation, and from that set collect only those Constraints that are Preconditions. In practice it is unlikely that ServiceOperations will have Constraints that are neither Preconditions nor Postconditions.

```
[precondition = constraint->collect  
    (c | c.ocIsKindOf(precondition))]
```

Constraint PostconditionDerivation:

The postcondition property is derived by using the constraint property inherited from ISO20022's RepositoryConcept element. The derivation rule is: start with the set of all Constraints that pertain to the ServiceOperation, and from that set collect only those that are Postconditions. In practice it is unlikely that ServiceOperations will have Constraints that are neither Preconditions nor Postconditions.

```
[postcondition = constraint->collect  
    (c | c.ocIsKindOf(postcondition))]
```

Constraint OutputMessageAlignment:

If a ServiceOperation has an output message, then its invocationKind must be either REQUEST\_RESPONSE or REQUEST\_DELAYED\_RESPONSE.

```
[outputMessage->notEmpty( ) implies  
(  
    invocationKind = REQUEST_RESPONSE or invocationKind =  
    REQUEST_DELAYED_RESPONSE  
)  
]
```

### Class ServiceRole

```
package BIAN::Level1
```

The unique business role that the owning Service Domain plays in the service landscape. ServiceRole is a specialization of the ISO 20022 Metamodel's BusinessRole element.

### Direct Superclasses

[BIANElement](#), [BusinessRole](#)

### Associations



: [ServiceDomain](#) [1]

### Attributes

businessPurpose : string [1]

A description of the ServiceDomain's unique business purpose,

### Constraints

## Enumeration InvocationKind

An enumeration of the four styles of ServiceOperation invocation. For information on the correspondence between these invocation styles and the interactions among Participants in BusinessScenarios, see the documentation of [BIANMessageTransmission](#).

```
package BIAN::Level2
```

### Literals

⦿ REQUEST\_RESPONSE

The client sends a request and stops working until it receives a response.

⦿ REQUEST\_DELAYED\_RESPONSE

The client makes a request and continues working, monitoring in some fashion for the eventual (i.e. "delayed") response.

⦿ HAND\_OFF\_REQUEST

The client makes a request without expecting a response.

⦿ NOTIFICATION

The operation is not triggered by an explicit request. Instead, the client sends the information in the input message to any ServiceDomain that has registered interest in being notified via the operation.

## B1. Capabilities-Responsibilities-Delegation Viewpoint

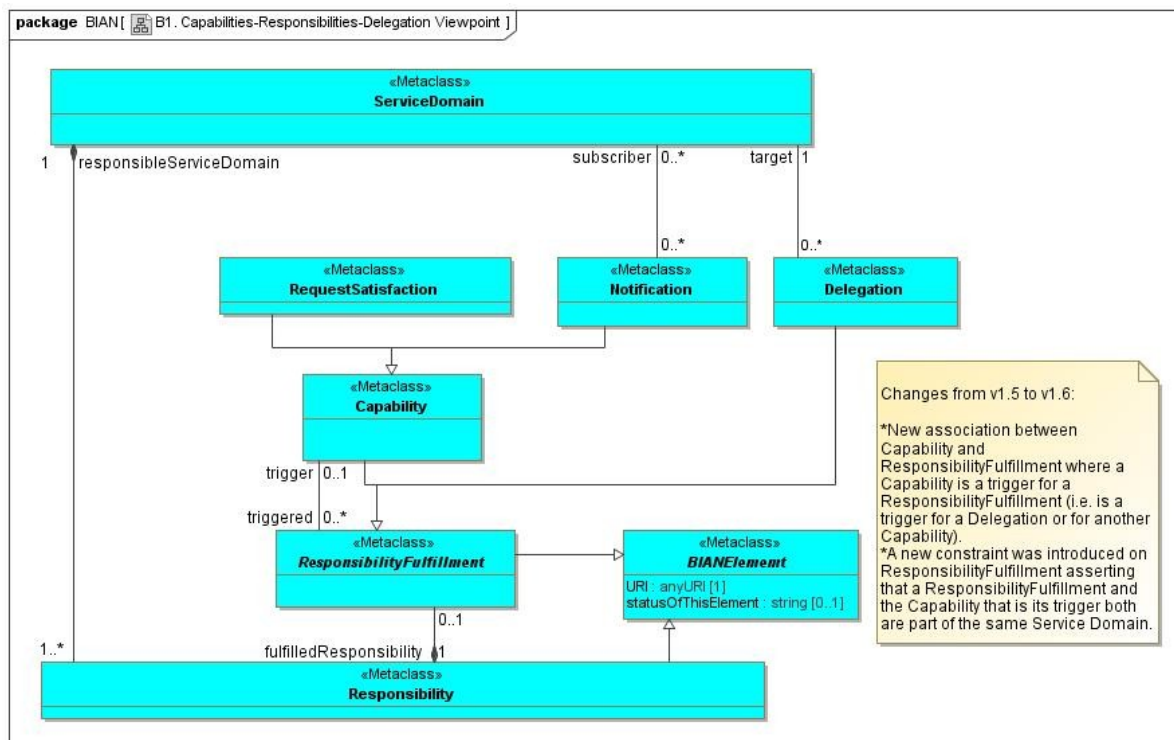


Figure 3. B1. Capabilities-Responsibilities-Delegation Viewpoint

This diagram shows the elements of the metamodel that support the definition of a ServiceDomain at Level 1.

A ServiceDomain owns a set of Responsibilities. Each Responsibility is fulfilled by means of a native Capability of the ServiceDomain or by a Delegation to a target ServiceDomain.

A native Capability can either be handled by publishing Notifications to subscriber ServiceDomains or by enabling service consumers to issue a request for service that the ServiceDomain commits to satisfying.

ServiceDomain also has a focusObject property, which points to a BusinessObject. The lifecycle of the focusObject is under the unique control of the ServiceDomain. A ServiceDomain also can have referencedObjects, which may be focusObjects for other ServiceDomains.

### Class Capability

```
package BIAN::Level1
```

A Capability is the capacity of a ServiceDomain to fulfill a Responsibility that it owns without relying on other Service Domains.

#### Direct Superclasses ResponsibilityFulfillment

#### Associations

✓ realizingOperation : [ServiceOperation](#) [0..\*]

The ServiceOperations that realize the Capability.

✓ triggered : [ResponsibilityFulfillment](#) [0..\*]

The ResponsibilityFulfillments -- i.e. the Capabilities and Delegations -- that the Capability triggers.

### Attributes

### Constraints

## Class Delegation

package BIAN::Level1

A Delegation is the fulfillment of a Responsibility owned by a ServiceDomain, achieved by enlisting the aid of another ServiceDomain.

### Direct Superclasses

[ResponsibilityFulfillment](#)

### Associations

✓ consumedOperation : [ServiceOperation](#) [1..\*]

The ServiceOperations that the ServiceDomain consumes in order to Delegate the fulfillment of a Responsibility.

✓ target : [ServiceDomain](#) [1]

The Service Domain to which the responsibility fulfillment is delegated.

### Attributes

### Constraints

Constraint DelegationToDifferentServiceDomain:

The target ServiceDomain of the Delegation is a different ServiceDomain than the ServiceDomain that owns the Responsibility that the Delegation fulfills.

[target <> fulfilledResponsibility.responsibleServiceDomain]

Constraint TriggerIsInSameServiceDomain:

A Delegation and the Capability that is its trigger both are part of the same Service Domain.

[not trigger.oclIsUndefined implies

(

trigger.fulfilledResponsibility.responsibleServiceDomain =



## BIAN Metamodel Specification

```
        self.fulfilledResponsibility.responsibleServiceDomain
    )
]
```

Constraint GiveItAName:

In English.

[]

### Class Notification


package BIAN::Level1

A Notification is a Capability to fulfill a Responsibility, which is carried out by publishing information to ServiceDomains that subscribe to receive the Notifications.

#### *Direct Superclasses*

[Capability](#)

#### *Associations*

 subscriber : [ServiceDomain](#) [0..\*]

The ServiceDomains that subscribe to receive the Notification.

#### *Attributes*

#### *Constraints*

Constraint InvocationKindNOTIFICATION:

The ServiceOperations that realize a Notification must have InvocationKind NOTIFICATION.

```
[realizingOperation->forAll
(
    operation | operation.invocationKind = InvocationKind::NOTIFICATION
)
]
```

### Class RequestSatisfaction

package BIAN::Level1

RequestSatisfaction is a Capability of a ServiceDomain to fulfill a Responsibility, by executing requests from service consumers.

#### *Direct Superclasses*

[Capability](#)

### Associations

### Attributes

### Constraints

Constraint InvocationKindNotNOTIFICATION:

The ServiceOperations that realize a Notification must not have InvocationKind NOTIFICATION.

```
[realizingOperation->forAll
(
  operation | not (operation.invocationKind =
InvocationKind::NOTIFICATION)
)
]
```

## Class Responsibility

package BIAN::Level1

A Responsibility is a commitment by a ServiceDomain to ensure the achievement of a business purpose.

A ServiceDomain can fulfill a Responsibility by means of one of its own Capabilities or by Delegation to other ServiceDomains.


### Direct Superclasses

[BIANElement](#), [RepositoryConcept](#)

### Associations

 responsibleServiceDomain : [ServiceDomain](#) [1]

The ServiceDomain that assumes the Responsibility. The association is a composition in which ServiceDomain plays the role of composite and Responsibility plays the role of component.

 responsibilityFulfillment : [ResponsibilityFulfillment](#) [0..1]

The native Capability, Delegation, or Notification that fulfills the Responsibility. The association is a composition in which Responsibility plays the role of composite and ResponsibilityFulfillment plays the role of component.

### Attributes

### Constraints

## Class ResponsibilityFulfillment

package BIAN::Level1

ResponsibilityFulfillment is the carrying out of a Responsibility of a ServiceDomain.

### *Direct Superclasses*

[BIANElement](#), [RepositoryConcept](#)

### *Associations*



fulfilledResponsibility : [Responsibility](#) [1]

The Responsibility being fulfilled.



trigger : [Capability](#) [0..1]

The Capability (if any) that triggers the ResponsibilityFulfillment, i.e. that triggers the Capability or Delegation.

### *Attributes*

### *Constraints*

Constraint TriggerIsInSameServiceDomain:

A Capability that triggers this ResponsibilityFulfillment must belong to the same ServiceDomain as this ResponsibilityFulfillment.

```
[trigger.fulfilledResponsibility.responsibleServiceDomain =  
self.fulfilledResponsibility.responsibleServiceDomain]
```

## B2. Capabilities-Responsibilities-Delegation with Level 2

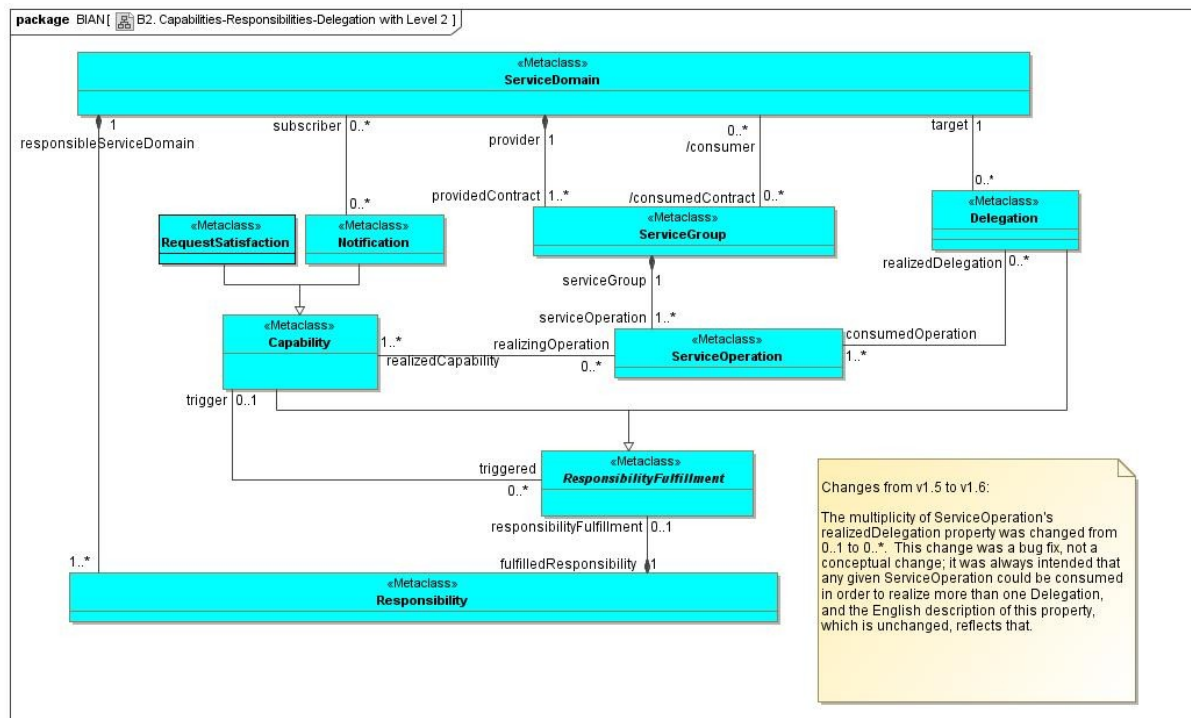


Figure 4. B2. Capabilities-Responsibilities-Delegation with Level 2

This diagram shows the relationships of Capabilities, Responsibilities, and Delegations -- which are defined at Level 1 -- to ServiceOperations, which are defined at Level 2. The Level 1 metaclasses are color-coded white and the Level 2 metaclasses are brown.

The fulfillment of a Responsibility via a native Capability of a ServiceDomain is realized via ServiceOperations that belong to a ServiceGroup that is owned by the ServiceDomain. Capability's realizingOperation property is designed to point to these ServiceOperations. The restriction that the realizingOperations must be owned by the same ServiceDomain is captured in a constraint on the Capability metaclass.

The fulfillment of a Responsibility via a Delegation to another ServiceDomain is realized via ServiceOperations that belong to a ServiceGroup that is owned by another ServiceDomain.

Delegation's consumingOperation property is designed to point to these ServiceOperations, and Delegation's 'target' property is designed to point to the ServiceDomain to which the Responsibility is delegated. The restriction that the consumingOperations must be owned by a different ServiceDomain is captured in a constraint owned by the Delegation metaclass.

## C. Business Object Viewpoint

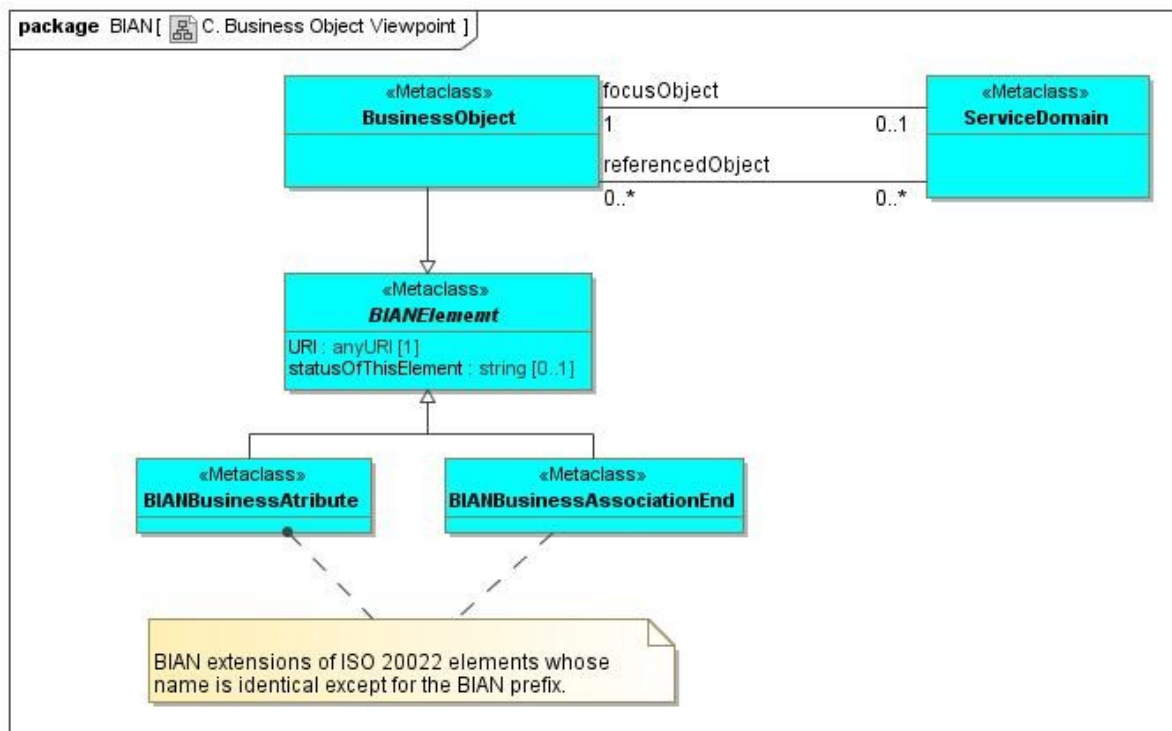


Figure 5. C. Business Object Viewpoint

This diagram shows the elements of the metamodel that pertain to BusinessObjects. BusinessObjects play a fundamental role in the definition of ServiceDomains, as each ServiceDomain has one focusObject whose instances it uniquely owns.

Much of the internal structure of a BusinessObject is not visible on this diagram because that structure is defined by the ISO20022 Metamodel, of which the BIAN Metamodel is an extension. The relationships between the BIAN and ISO20022 Metamodels are shown in the diagram named "C. BIAN-ISO20022 Business Object Viewpoint" in the BIAN\_ISO2000 package. (The BIAN\_ISO20022 package focuses on how the BIAN Metamodel extends the ISO20022 Metamodel). That diagram reveals the following:

- A BusinessObject is a specialization of an ISO20022 BusinessComponent.
- A BIANBusinessAttribute is a specialization of an ISO20022 BusinessAttribute. There is a similar relationship between BIANBusinessAssociationEnd and ISO20022 BusinessAssociationEnd.

The relationships between BusinessObjects and BIANMessageDefinitions are also not visible on this diagram, because those relationships are defined by the ISO20022 Metamodel where it defines the relationships between BusinessComponents and MessageComponents.

### Class BIANBusinessAssociationEnd

package BIAN::Level1

BIANBusinessAssociationEnd is a specialization of ISO20022 BusinessAssociationEnd. An ISO20022 BusinessAssociationEnd is one end of an association between two BusinessComponents.

What makes BIANBusinessAssociationEnd different from ISO20022 BusinessAssociationEnd is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties defined from those superclasses.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 BusinessAssociationEnd obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

### Direct Superclasses

[BIANElement](#), [BusinessAssociationEnd](#), [TaggableElement](#)

### Associations

### Attributes

### Constraints

## Class BIANBusinessAttribute

```
package BIAN::Level1
```

BIANBusinessAttribute is a specialization of ISO20022 BusinessAttribute. An ISO20022 BusinessAttribute is an element of a BusinessComponent, such as *StartDate*.

What makes BIANBusinessAttribute different from ISO20022 BusinessAttribute is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties from those two superclasses.

Another distinction is that an ISO20022 BusinessAttribute can have a simple type or a complex type, whereas a BIANBusinessAttribute can only have a simple type.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 BusinessAttribute obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

### Direct Superclasses

[BIANElement](#), [BusinessAttribute](#), [TaggableElement](#)

### Associations

### Attributes

### Constraints

Constraint SimpleTypesOnly:

A BIANBusinessAttribute can only have a simple type. It is not allowed to have a complex type. An ISO20022 BusinessAttribute is allowed to have either a complex type or a simple type, so a BIANBusinessAttribute is more tightly constrained.

### Comment

The reason for this restriction is that allowing an attribute to have a complex type -- that is, allowing the type of an attribute to be a full-blown BusinessObject -- brings with it some assumptions that can lead to a loss of semantic clarity.

Imagine that we have two BusinessObjects B1 and B2, and that B1 has an attribute whose type is B2. That is semantically equivalent to saying that there is an association between B1 and B2 that has the following characteristics:

1. The association is a composition in which B1 plays the role of the composite and B2 plays the role of component.
2. The cardinality on the B1 side of the composition association is exactly 1.

The problem with these two assumptions is that often at least one of them is not true of associations between two complex types. We thus prefer to force the modeler to make explicit decisions as to whether the association is a composition and as to the cardinality on both sides of the association.

```
[complexType->isEmpty()  
]
```

### Class BusinessObject

```
package BIAN::Level1
```

A BusinessObject is an individually distinguishable element characterized by the fact that it has a well defined identity, structure and behavior.

BusinessObject is a specialization of ISO20022 BusinessComponent. What makes BusinessObject different from ISO20022 BusinessComponent is that it has additional properties, including its associations with ServiceDomain and the properties that it inherits from BIANElement and TaggableElement.

BIAN BusinessObjects are fundamental to the definition of ServiceDomains, as each ServiceDomain has exactly one focus object, and possibly multiple other objects that the ServiceDomain references.

#### Comments

- o In UML the term object denotes an instance of a class. However, in daily IT language, and also within BIAN, the term object is used synonymously with the formal meaning of class. The ISO20022 term "BusinessComponent" sidesteps the object vs. class issue.
- o The BIAN object models are at the conceptual level and are made from a business or application perspective.
- o The object replaces the traditional notion of entity (as in entity relationship diagrams). The object notion is somewhat more precise than the entity notion, but both concepts have an equally broad scope. They can be used to model "anything of interest" in the system being studied.

#### Direct Superclasses

[BIANElement](#), [BusinessComponent](#), [TaggableElement](#)

#### Associations

```
serviceDomainForFocusObject : ServiceDomain [0..1]
```

The ServiceDomain for which this is the focusObject.

```
serviceDomainForReferencedObject : ServiceDomain [0..*]
```

The ServiceDomains for which this is a referencedObject.

## D. Message Viewpoint

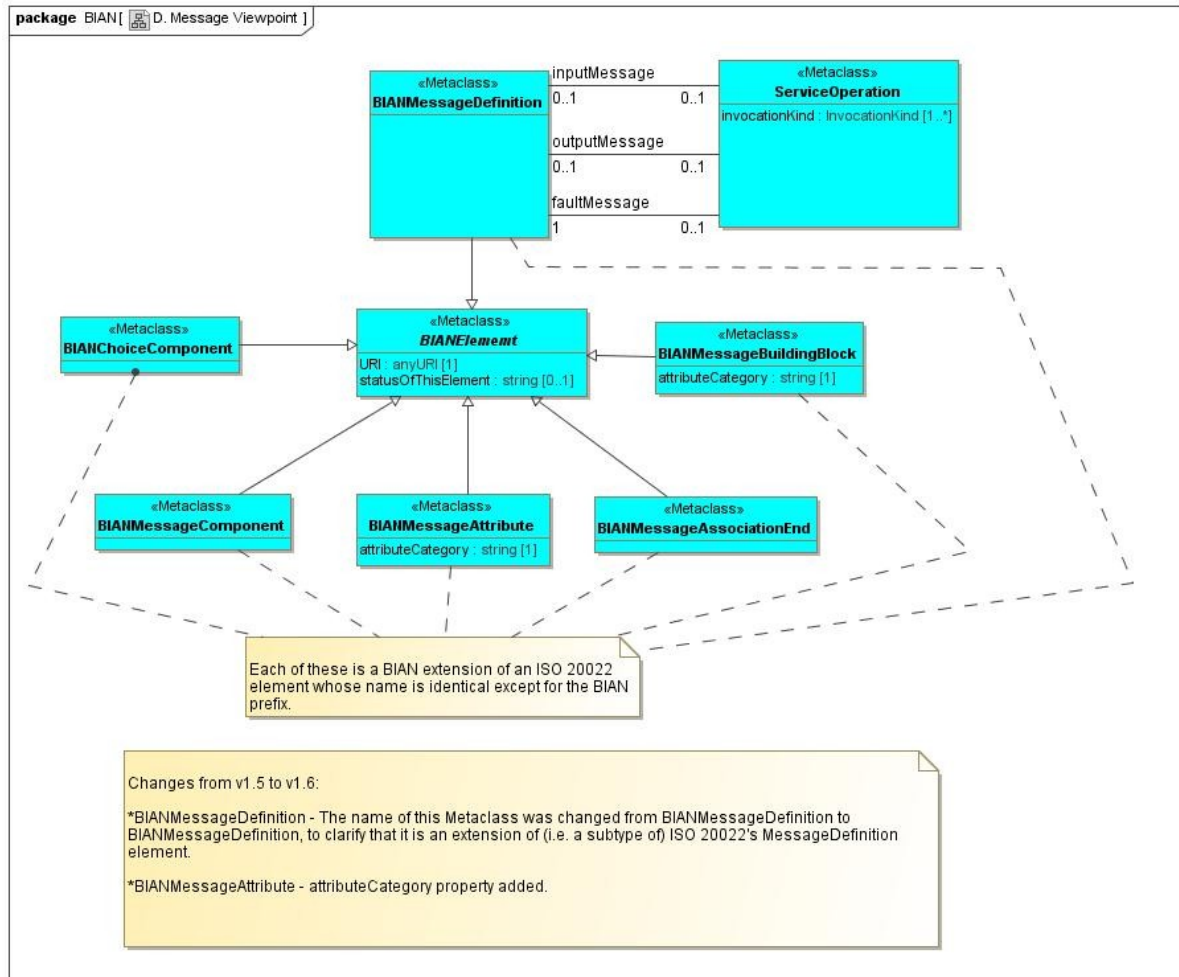


Figure 6. D. Message Viewpoint

This diagram shows the elements of the metamodel that pertain to messages. BIANMessageDefinitions play a fundamental role in the definition of ServiceOperations, as a ServiceOperation may have an input message, may have an output message, and must have a fault message.

Much of the internal structure of a BIANMessageDefinition is not visible on this diagram because that structure is defined by the ISO20022 Metamodel, of which the BIAN Metamodel is an extension. The diagram named "D. BIAN-ISO20022 Message Viewpoint" in the BIAN\_ISO2000 package defines the relationships between the BIAN and ISO20022 elements that pertain to messages. (The BIAN\_ISO20022 package focuses on how the BIAN Metamodel extends the ISO20022 Metamodel). That diagram reveals the following:

- BIANMessageDefinition is a specialization of ISO20022 MessageDefinition
- BIANMessageComponent is a specialization of ISO20022 MessageComponent.
- BIANMessageAttribute is a specialization of ISO20022 MessageAttribute. There is a similar relationship between BIANMessageAssociationEnd and ISO20022 MessageAssociationEnd.B



- BIANMessageBuildingBlock is a specialization of ISO20022 MessageBuildingBlock. In ISO20022, a MessageBuildingBlock defines how a MessageComponent or DataType is used in a MessageDefinition.

The relationships between BusinessObjects and BIANMessageDefinitions are also not visible on this diagram, because those relationships are defined by the ISO 20022 Metamodel where it defines the relationships between BusinessComponents and MessageComponents.

### Class BIANChoiceComponent

```
package BIAN::Level2
```

BIANChoiceComponent is a specialization of ISO20022 ChoiceComponent. An ISO20022 ChoiceComponent is a set of MessageElements from which one MessageElement is selected when instantiating the ChoiceComponent. A ChoiceComponent is conceptually similar to a union in a traditional programming language or a choice element in XML Schema.

What makes BIANChoiceComponent different from ISO20022 ChoiceComponent is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties from those two superclasses.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 ChoiceComponent obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

#### *Direct Superclasses*

[BIANElement](#), [ChoiceComponent](#), [TaggableElement](#)

#### *Associations*

#### *Attributes*

#### *Constraints*

### Class BIANMessageAssociationEnd

```
package BIAN::Level2
```

BIANMessageAssociationEnd is a specialization of ISO20022 MessageAssociationEnd. An ISO20022 MessageAssociationEnd is one end of a relationship between two MessageComponentTypes .

What makes BIANMessageAssociationEnd different from ISO20022 MessageAssociationEnd is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties from those two superclasses.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 MessageAssociationEnd obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

#### *Direct Superclasses*

[BIANElement](#), [MessageAssociationEnd](#), [TaggableElement](#)

### Associations

### Attributes

### Constraints

## Class BIANMessageAttribute

```
package BIAN::Level2
```

BIANMessageAttribute is a specialization of ISO20022 MessageAttribute. In ISO20022, a MessageAttribute is an element of a MessageComponent, such as *StartDate*, which can optionally be traceable back to a BusinessComponent or to an element of a BusinessComponent. A MessageAttribute's type typically is a simple type but can be a complex type.

What makes BIANMessageAttribute different from ISO20022 MessageAttribute is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties from those two superclasses.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 MessageAttribute obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

### Direct Superclasses

[BIANElement](#), [MessageAttribute](#), [TaggableElement](#)

### Associations

### Attributes

```
attributeCategory : string [1]
```

The category of attributes into which the BIANMessageAttribute falls. At this stage BIAN is not hard-wiring the categories themselves into an enumeration, which is why the type is string, but as of this writing the categories in use are: identifier, depiction, status, control, session, analysis, view.

### Constraints

## Class BIANMessageBuildingBlock

```
package BIAN::Level2
```

BIANMessageBuildingBlock is a specialization of ISO20022 MessageBuildingBlock. An ISO20022 MessageBuildingBlock defines how a MessageComponent or DataType is assembled into a BIANMessageDefinition.

What makes BIANMessageBuildingBlock different from ISO20022 MessageBuildingBlock is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties from those two superclasses.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 MessageBuildingBlock obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

### *Direct Superclasses*

[BIANElement](#), [MessageBuildingBlock](#), [TaggableElement](#)

### *Associations*

### *Attributes*

```
attributeCategory : string [1]
```

The category of attributes into which the BIANMessageBuildingBlock falls. At this stage BIAN is not hard-wiring the categories themselves into an enumeration, which is why the type is string, but as of this writing the categories in use are: identifier, depiction, status, control, session, analysis, view.

### *Constraints*

## **Class BIANMessageComponent**

```
package BIAN::Level2
```

BIANMessageComponent is a specialization of ISO20022 MessageComponent. An ISO20022 MessageComponent is a set of MessageElements that can be used as a building block when assembling a BIANMessageDefinition.

What makes BIANMessageComponent different from ISO20022 MessageComponent is that it is a subclass of BIANElement and a subclass of TaggableElement, and thus inherits additional properties from those two superclasses.

"BIAN" was added to the name of this class in order to make its distinction from ISO20022 MessageComponent obvious, but it was not strictly necessary for disambiguation since two UML classes can have the same unqualified name as long as they are in separate packages.

### *Direct Superclasses*

[BIANElement](#), [MessageComponent](#), [TaggableElement](#)

### *Associations*

### *Attributes*

### *Constraints*

## **Class BIANMessageDefinition**

```
package BIAN::Level2
```

A BIANMessageDefinition is a structured set of information exchanged in the context of a ServiceOperation. A BIANMessageDefinition is assembled from MessageComponents.

### **Comment**

BIANMessageDefinition is a subtype of ISO 20022's MessageDefinition element. ISO 20022 uses the term "MessageDefinition" instead of "BIANMessageDefinition" for its element because what UML calls

a "BIANMessageDefinition" is what ISO 20022 calls a "MessageTransmission." When we refer simply to a "message" in this discussion we mean an instance of BIANMessageDefinition.)

### Direct Superclasses

[BIANElement](#), [MessageDefinition](#), [TaggableElement](#)

### Associations

✍ opForInputMessage : [ServiceOperation](#) [0..1]

The ServiceOperations for which this BIANMessageDefinition is the input parameter.

✍ opForOutputMessage : [ServiceOperation](#) [0..1]

The ServiceOperations for which this BIANMessageDefinition is the output parameter.

✍ opForFaultMessage : [ServiceOperation](#) [0..1]

The ServiceOperations for which this BIANMessageDefinition is the fault BIANMessageDefinition.

### Attributes

### Constraints

Constraint InputOutputFaultXOR:

Exactly one of opForInput, opForOutput, and opForFault is not empty. In other words a BIANMessageDefinition must be either the input message for a ServiceOperation, the output message for a service operation, or the fault message for a service operation.

```
[opForInput->size( ) + opForOutput->size( ) + opForFault->size( ) = 1  
]
```

## E. Scenario Viewpoint

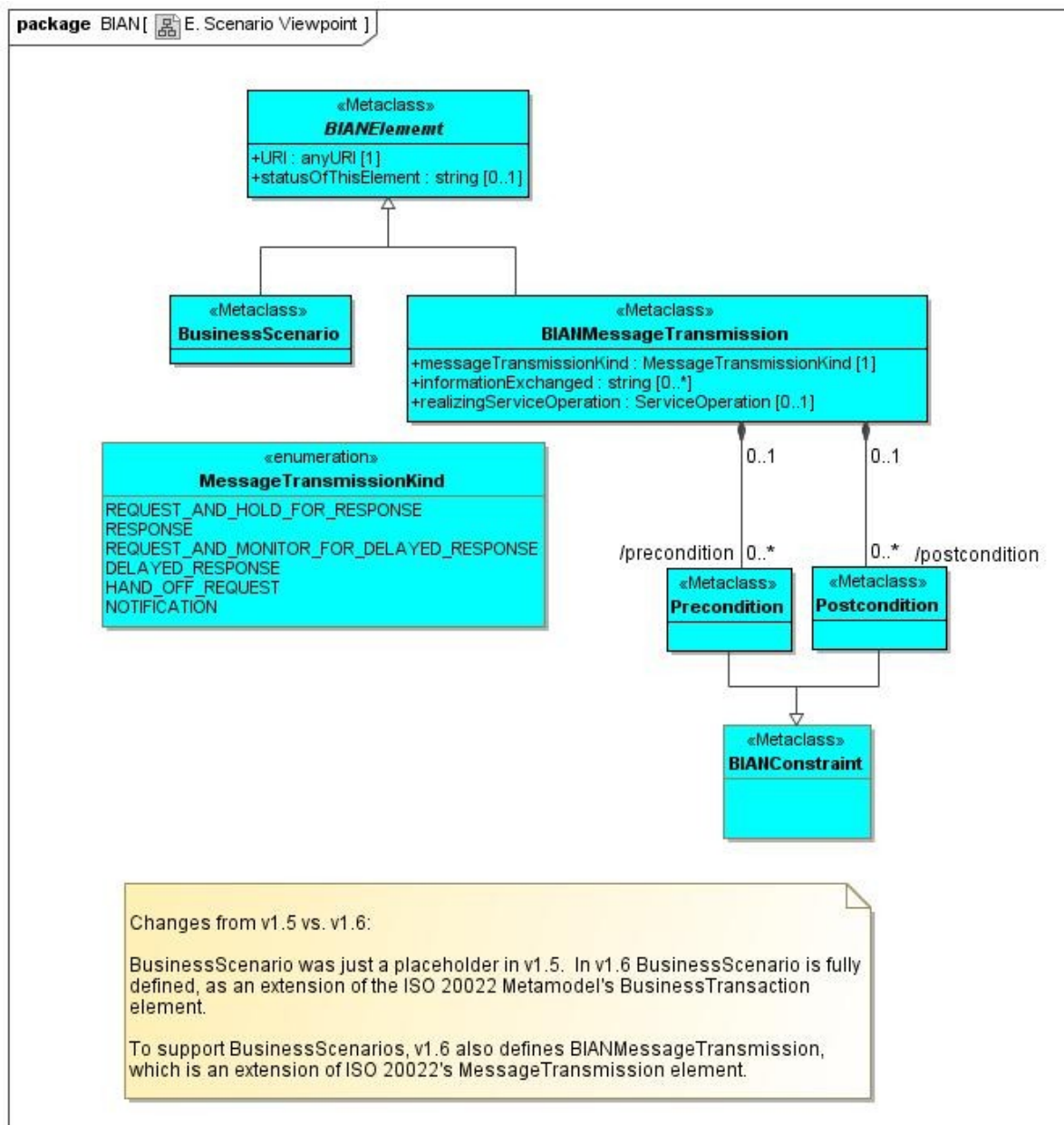


Figure 7. E. Scenario Viewpoint

This diagram shows the elements that BIAN defines to model BusinessScenarios. The reason that there are only two metaclasses to support BusinessScenarios in all their richness is that the BIAN Metamodel heavily reuses the ISO 20022 Metamodel of BusinessTransaction and MessageTransmission, which BusinessScenario and BIANMessageTransmission subtype, respectively. See the BIAN\_ISO20022 Package for detail about this reuse of ISO 20022, as explained in the documentation of the BIAN-ISO20022 Message Viewpoint diagram; it is vital to understand this aspect of the reuse of ISO 20022 in order to understand how BIAN models BusinessScenarios.

BIAN does not publish business scenarios as canonical standards. It models business scenarios in order to discover, validate, and explain the canonical definitions of ServiceDomains and their elements. But the scenarios themselves are not canonical, because a bank that uses BIAN must have flexibility in how it combines the ServiceDomains to execute its business.

## Class BIANMessageTransmission

```
package BIAN::Level1
```

A BIANMessageTransmission represents the sending of a communication between two Participants in a BusinessScenario. The word "BIAN" is part of the name of this element in order to distinguish this element from the MessageTransmission element in ISO 20022, which BIANMessageTransmission specializes. The specialization is necessary in order to add the extra metadata contained in values of the messageTransmissionKind.

The six enumerated messageTransmissionKind values support four different modes of communication among the Participants in a Business Scenario:

1. The case where the sender makes a request and holds for a response is modeled as two BIANMessageTransmissions, one for sending the request and one for sending the response. The BIANMessageTransmission corresponding to the request has REQUEST\_AND\_HOLD\_FOR\_RESPONSE as the value of the messageTransmissionKind, whereas RESPONSE is the value for the BIANMessageTransmission corresponding to the RESPONSE. The sender for the first transmission is the receiver for the second, and vice versa.
2. The case where the sender makes a request and continues working, monitoring in some fashion for the eventual (i.e. "delayed") response is also modeled as two BIANMessageTransmissions for which the messageTransmissionKind values are REQUEST\_AND\_MONITOR\_FOR\_RESPONSE and DELAYED\_RESPONSE for sending and responding, respectively. As for case 1, the sender for the first transmission is the receiver for the second, and vice versa.
3. The case where the sender makes a request without expecting a response is modeled as one BIANMessageTransmission whose messageTransmissionKind value is HAND\_OFF\_REQUEST.
4. The case where the sender sends a notification to potentially multiple parties is modeled as one BIANMessageTransmission per recipient. The messageTransmissionKind value is NOTIFICATION for all of the BIANMessageTransmissions

These four cases correspond to the four possible values of the invocationKind property of ServiceOperation. Those four values are defined in the [InvocationKind](#) enumeration. The reason that there are four InvocationKind values and six MessageTransmissionKind values is that the InvocationKind values REQUEST\_RESPONSE and REQUEST\_DELAYED\_RESPONSE imply the transmission of a request and the transmission of a response, and we have explicit MessageTransmissionKind values for the request and the response. The correspondence is as follows:

<u>MessageTransmissionKind Value</u>	<u>InvocationKind Value</u>
REQUEST_AND_HOLD_FOR_RESPONSE	REQUEST_RESPONSE
RESPONSE	REQUEST_RESPONSE
REQUEST_AND_MONITOR_FOR_RESPONSE	REQUEST_DELAYED_RESPONSE
DELAYED_RESPONSE	REQUEST_DELAYED_RESPONSE
HAND_OFF_REQUEST	HAND_OFF_REQUEST
NOTIFICATION	NOTIFICATION

The definition of this element, BIANMessageTransmission, includes constraints written in OCL and English that enforce this correspondence. To understand how these constraints are written, consider that a BIANMessageTransmission can be specified simply as a communication between two Participants in a BusinessScenario, without formally specifying the data that would be transmitted. At a later time, a BIANMessageDefinition can be bound to the BIANMessageTransmission. This fully

conforms to the ISO 20022 Metamodel, which the BIAN Metamodel extends; in ISO 20022, a MessageTransmission may or may not be bound to a MessageDefinition.

(BIANMessageDefinition is a subtype of ISO 20022's MessageDefinition element. ISO 20022 uses the term "MessageDefinition" instead of "BIANMessageDefinition" for its element because what UML calls a "BIANMessageDefinition" is what ISO 20022 calls a "MessageTransmission." When we refer simply to a "message" in this discussion we mean an instance of BIANMessageDefinition.)

A BIANMessageDefinition can be specified as the type of a parameter of a ServiceOperation, playing the role either of an input message, output message, or fault message for the operation. Thus, when a BIANMessageDefinition is bound to a BIANMessageTransmission, an indirect relationship is established between the message transmission and the operation for which the message is the input or output message. The associations in the Metamodel allow us to write OCL that formalizes the correspondence between the MessageTransmissionKind and InvocationKind values in the above table.

For instance, one constraint says that a BIANMessageDefinition bound to a REQUEST\_AND\_MONITOR\_FOR\_RESPONSE message transmission must be the input message for an operation whose InvocationKind is REQUEST\_DELAYED\_RESPONSE, while another says that a message bound to a DELAYED\_RESPONSE message transmission must be the output message for an operation whose InvocationKind is REQUEST\_RESPONSE.

### Direct Superclasses

[BIANElement](#), [MessageTransmission](#)

### Associations



precondition : [Precondition](#) [0..\*]

The Preconditions that constrain this BIANMessageTransmission.

A Precondition of a BIANMessageTransmission specifies a condition of the system that must be when the transmission starts. It is distinct from a Postcondition, which must be true when the behavior triggered by the transmission finishes.

Precondition is a subclass of BIANConstraint.

### Comments

As we describe BIANMessageTransmission at the level of business semantics, the Preconditions are of a business nature.



postcondition : [Postcondition](#) [0..\*]

The Postconditions that constrain this BIANMessageTransmission.

A Postcondition of a BIANMessageTransmission specifies a condition of the system that must be true at the time of completion of the behavior that the transmission triggers. It is distinct from a Precondition, which must be true at the start of the transmission.

Postcondition is a subclass of BIANConstraint.

### Comments

As we describe BIANMessageTransmission at the level of business semantics, the Preconditions are of a business nature.

### Attributes



`messageTransmissionKind : MessageTransmissionKind [1]`

BusinessScenarios can include several different kinds of message transmission. This property specifies the specific kind. See the documentation of [BIANMessageTransmission](#) for a full explanation.

`informationExchanged : string [0..*]`

Informal specification of any information exchanged via the message transmission.

`realizingServiceOperation : ServiceOperation [0..1]`

**Case 1:** For a `BIANMessageTransmission` whose `messageTransmissionKind` is `REQUEST_AND_HOLD_FOR_RESPONSE`, `REQUEST_AND_MONITOR_FOR_DELAYED_RESPONSE`, `HAND_OFF_REQUEST`, or `NOTIFICATION`, the value of this attribute is a reference to the `ServiceOperation` whose execution is triggered by the `BIANMessageTransmission`. A value for this attribute need only be provided in cases where the `ServiceOperation` has no input message.

**Case 2:** For a `BIANMessageTransmission` whose `messageTransmissionKind` is `RESPONSE` or `DELAYED_RESPONSE`, the value of this attribute is a reference to the `ServiceOperation` whose response is conveyed via the `BIANMessageTransmission`. A value for this attribute need only be provided in cases where the `Service Operation` has no output message.

A value for this attribute need only appear when there is no `BIANMessageDefinition`.

### Comment

Typically we bind an input message or an output message of a `ServiceOperation` to a `BIANMessageTransmission`. That indirectly ties the `ServiceOperation` to the `BIANMessageTransmission`.

However, in Case 1 above, if the `ServiceOperation` whose invocation is triggered by the `BIANMessageTransmission` has no input message, then populating this attribute provides the only way to express the connection between the `BIANMessageTransmission` and the `ServiceOperation`.

Similarly, in Case 2 above, if the `ServiceOperation` whose response is carried by the `BIANMessageTransmission` has no output message, then populating this attribute provides the only way to express the connection between the `BIANMessageTransmission` and the `ServiceOperation`.

### Constraints

Constraint `RequestAndHoldAlignmentWithOperation`:

A message that is bound to a `REQUEST_AND_HOLD_FOR_RESPONSE` message transmission must be the input message for a service operation whose `invocationKind` is `REQUEST_RESPONSE`.

```
[ (messageTransmissionKind =  
MessageTransmissionKind::REQUEST_AND_HOLD_FOR_RESPONSE and derivation-  
>notEmpty() ) implies  
(  
    derivation->forAll  
    (  
        messageTypeTrace |  
messageTypeTrace.messageDefinition.oclAsType (BIANMessageDefinition) .opForIn  
putMessage->notEmpty() and  
  
messageTypeTrace.messageDefinition.oclAsType (BIANMessageDefinition) .opForIn  
putMessage.invocationKind = REQUEST_RESPONSE  
    )  
) ]
```



```
)  
]
```

Constraint ResponseAlignmentWithOperation:

**A message that is bound to a RESPONSE message transmission must be the output message for a service operation whose invocationKind is REQUEST\_RESPONSE.**

```
[(messageTransmissionKind = MessageTransmissionKind::RESPONSE and  
derivation->notEmpty() ) implies  
(  
    derivation->forAll  
    (  
        messageTypeTrace |  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForOu  
tputMessage->notEmpty() and  
  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForOu  
tputMessage.invocationKind = REQUEST_RESPONSE  
    )  
)  
]
```

Constraint RequestAndMonitorAlignmentWithOperation:

**A message that is bound to a REQUEST\_AND\_MONITOR\_FOR\_DELAYED\_RESPONSE message transmission must be the input message for a service operation whose invocationKind is REQUEST\_DELAYED\_RESPONSE.**

```
[(messageTransmissionKind =  
MessageTransmissionKind::REQUEST_AND_MONITOR_FOR_DELAYED_RESPONSE and  
derivation->notEmpty() ) implies  
(  
    derivation->forAll  
    (  
        messageTypeTrace |  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForIn  
putMessage->notEmpty() and  
  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForIn  
putMessage.invocationKind = REQUEST_DELAYED_RESPONSE  
    )  
)  
]
```

Constraint NotificationAlignmentWithOperation:

**A message that is bound to a NOTIFICATION message transmission must be the input message for a service operation whose invocationKind is NOTIFICATION.**

```
[(messageTransmissionKind = MessageTransmissionKind::NOTIFICATION and  
derivation->notEmpty() ) implies  
(  
    derivation->forAll  
    (  
        messageTypeTrace |  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForIn  
putMessage->notEmpty() and  
  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForIn  
putMessage.invocationKind = NOTIFICATION  
    )  
)  
]
```

```
)  
)  
]
```

Constraint DelayedResponseAlignmentWithOperation:

A message that is bound to a DELAYED\_RESPONSE message transmission must be the output message for a service operation whose invocationKind is REQUEST\_DELAYED\_RESPONSE.

```
[(messageTransmissionKind = MessageTransmissionKind::DELAYED_RESPONSE and  
derivation->notEmpty() ) implies  
(  
    derivation->forall  
    (  
        messageTypeTrace |  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForOu  
tputMessage->notEmpty() and  
  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForOu  
tputMessage.invocationKind = REQUEST_DELAYED_RESPONSE  
    )  
)  
]
```

Constraint HandOffAlignmentWithOperation:

A message that is bound to a HAND\_OFF\_REQUEST message transmission must be the input message for a service operation whose invocationKind is HAND\_OFF\_REQUEST.

```
[(messageTransmissionKind = MessageTransmissionKind::HAND_OFF_REQUEST and  
derivation->notEmpty() ) implies  
(  
    derivation->forall  
    (  
        messageTypeTrace |  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForIn  
putMessage->notEmpty() and  
  
messageTypeTrace.messageDefinition.oclAsType(BIANMessageDefinition).opForIn  
putMessage.invocationKind = HAND_OFF_REQUEST  
    )  
)  
]
```

Constraint BindToAtMostOneMessage:

A BIANMessageTransmission can be bound to at most one BIANMessageDefinition. (In ISO 20022, a MessageTransmission can be bound to zero to many MessageDefinitions, so this tightens the cardinality constraint. Note that BIANMessageDefinition is a subtype of ISO 20022 MessageDefinition.)

```
[derivation->size( ) <= 1]
```

Constraint PostconditionDerivation:

The postcondition property is derived by using the constraint property inherited from ISO20022's RepositoryConcept element. The derivation rule is: start with the set of all Constraints that pertain to the BIANMessageTransmission and from that set collect only those that are Postconditions. In practice it is unlikely that BIANMessageTransmissions will have Constraints that are neither Preconditions nor Postconditions.

## BIAN Metamodel Specification

```
[postcondition = constraint->collect  
  (c | c.oclIsKindOf(postcondition))]
```

Constraint PreconditionDerivation:

The precondition property is derived by using the constraint property inherited from ISO20022's RepositoryConcept element. The derivation rule is: start with the set of all Constraints that pertain to the BIANMessageTransmission and from that set collect only those that are Preconditions. In practice it is unlikely that BIANMessageTransmissions will have Constraints that are neither Preconditions nor Postconditions.

```
[precondition = constraint->collect  
  (c | c.oclIsKindOf(precondition))]
```

Constraint AlignmentWithServiceDomain:

```
[]
```

### Class BusinessScenario

```
package BIAN::Level1
```

A BusinessScenario is a model of how some ServiceDomains might collaborate together in response to a business event.

BIAN does not publish business scenarios as canonical standards. It models business scenarios in order to discover, validate, and explain the canonical definitions of ServiceDomains and their elements. But the scenarios themselves are not canonical, because a bank that uses BIAN must have flexibility in how it combines the ServiceDomains to execute its business.

#### Comment

A BusinessScenario defines an archetypical flow. The sequence as modeled can change in practice. Some interactions between ServiceDomains might be obsolete and others might be missing.

In theory there is no practical limit to the number of BusinessScenarios and possible variations that could be assembled using the standard BIAN ServiceDomains. BIAN uses combinations of BusinessScenarios to clarify and explain the roles of, boundaries of, and exchanges among Service Domains.

It is not BIAN's aim and intention to provide a complete set of Business Scenarios. Business Scenarios are an aid to populate and one way to reference the contents of the Service Landscape.

#### Direct Superclasses

[BIANElement](#), [BusinessTransaction](#), [TaggableElement](#)

#### Associations

### Attributes

### Constraints

## Enumeration MessageTransmissionKind

The enumeration of the allowable kinds of BIANMessageTransmissions. See the documentation of [BIANMessageTransmission](#) for a full explanation.

```
package BIAN::Level1
```

### Literals

#### 🔹 REQUEST\_AND\_HOLD\_FOR\_RESPONSE

This kind of BIANMessageTransmission represents the transmission of a request in which the BusinessScenario Participant that sends the request holds (i.e. waits) for a response from the Participant that receives the request.

#### 🔹 RESPONSE

This kind of BIANMessageTransmission represents the transmission of a response to a request in which the BusinessScenario Participant that sends the request holds (i.e. waits) for the response from the Participant that receives the request. The sender of the response is the receiver of the request.

#### 🔹 REQUEST\_AND\_MONITOR\_FOR\_DELAYED\_RESPONSE

This kind of BIANMessageTransmission represents the transmission of a request in which the BusinessScenario Participant that sends the request continues working after sending the request and monitors for an eventual (i.e. "delayed") response, rather than holding (i.e. waiting) for a response from the Participant that receives the request.

#### 🔹 DELAYED\_RESPONSE

This kind of BIANMessageTransmission represents the transmission of a delayed response to a request in which the BusinessScenario Participant that sends the request continues working after sending the request and monitors for the eventual (i.e. "delayed") response from the Participant that receives the request, rather than holding (i.e. waiting) for a response. The sender of the response is the receiver of the request.

#### 🔹 HAND\_OFF\_REQUEST

This kind of BIANMessageTransmission represents the transmission of a request in which the BusinessScenario Participant that sends the request does not expect a response from the Participant that receives the request.

#### 🔹 NOTIFICATION

This kind of BIANMessageTransmission represents the transmission of a notification in which the BusinessScenario Participant that sends the request notifies the Participant that receives the request about some event and expects no response.

## F. Implementation Component Viewpoint

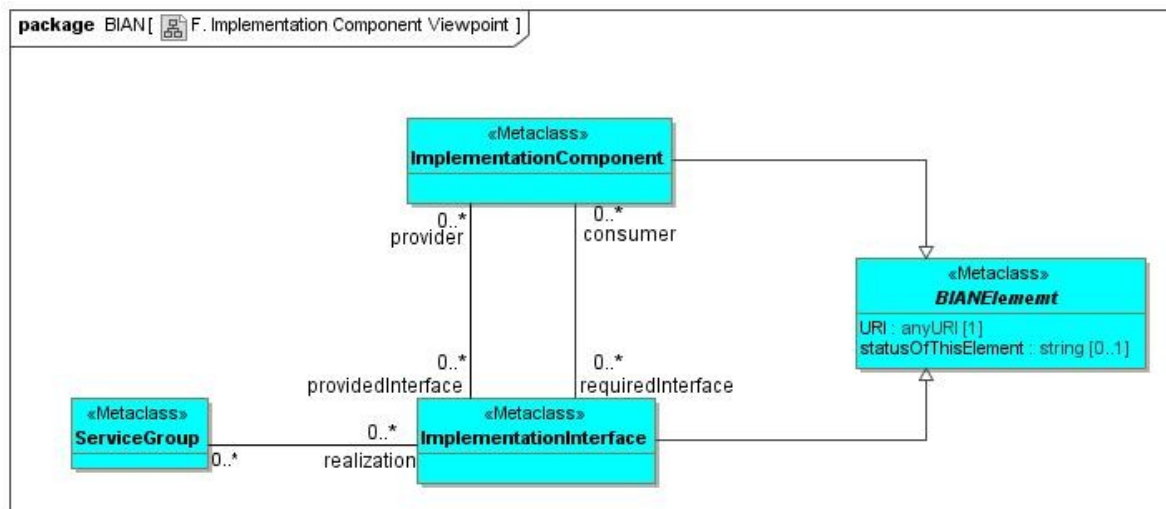


Figure 8. F. Implementation Component Viewpoint

This diagram highlights ImplementationComponents and ImplementationInterfaces, which are BIAN Level 3 elements.

An ImplementationInterface is a realization of some or all of the operations of a ServiceGroup. An ImplementationComponent provides ImplementationInterfaces that consumers can invoke, and requires the consumption of ImplementationInterfaces from other ImplementationComponents.

BIAN does not publish ImplementationComponents and ImplementationInterfaces as canonical standards. The realization of ServiceGroups is where implementors differentiate and compete in the marketplace, while adhering to the specifications of the ServiceGroups, which BIAN does publish as canonical standards.

BIAN may from time to time define ImplementationComponents and ImplementationInterfaces in the course of proof-of-concept exercises, but they will not be canonical.

### Class ImplementationComponent

```
package BIAN::Level3
```

An ImplementationComponent is a software component that provides interfaces to consumers and consumes the interfaces provided by other ImplementationComponents.

#### Direct Superclasses

[BIANElement](#), [TaggableElement](#)

#### Associations

```
providedInterface : ImplementationInterface [0..*]
```

The ImplementationInterfaces that the ImplementationComponent makes available to consumers.

```
requiredInterface : ImplementationInterface [0..*]
```

The ImplementationInterfaces that the ImplementationComponent must consume from other ImplementationComponents.

### Attributes

### Constraints

## Class ImplementationInterface

package BIAN::Level3

An ImplementationInterface is a software interface that implements one or more ServiceGroups.

### Direct Superclasses

[BIANElement](#), [TaggableElement](#)

### Associations

✍ serviceGroup : [ServiceGroup](#) [0..\*]

The ServiceGroups that the ImplementationInterface realizes.

✍ provider : [ImplementationComponent](#) [0..\*]

The ImplementationComponents that make the ImplementationInterface available to consumers.

✍ consumer : [ImplementationComponent](#) [0..\*]

The ImplementationComponents that consume the ImplementationInterface.

### Attributes

### Constraints

## G. Canonical Instances vs. Non-Canonical Instances

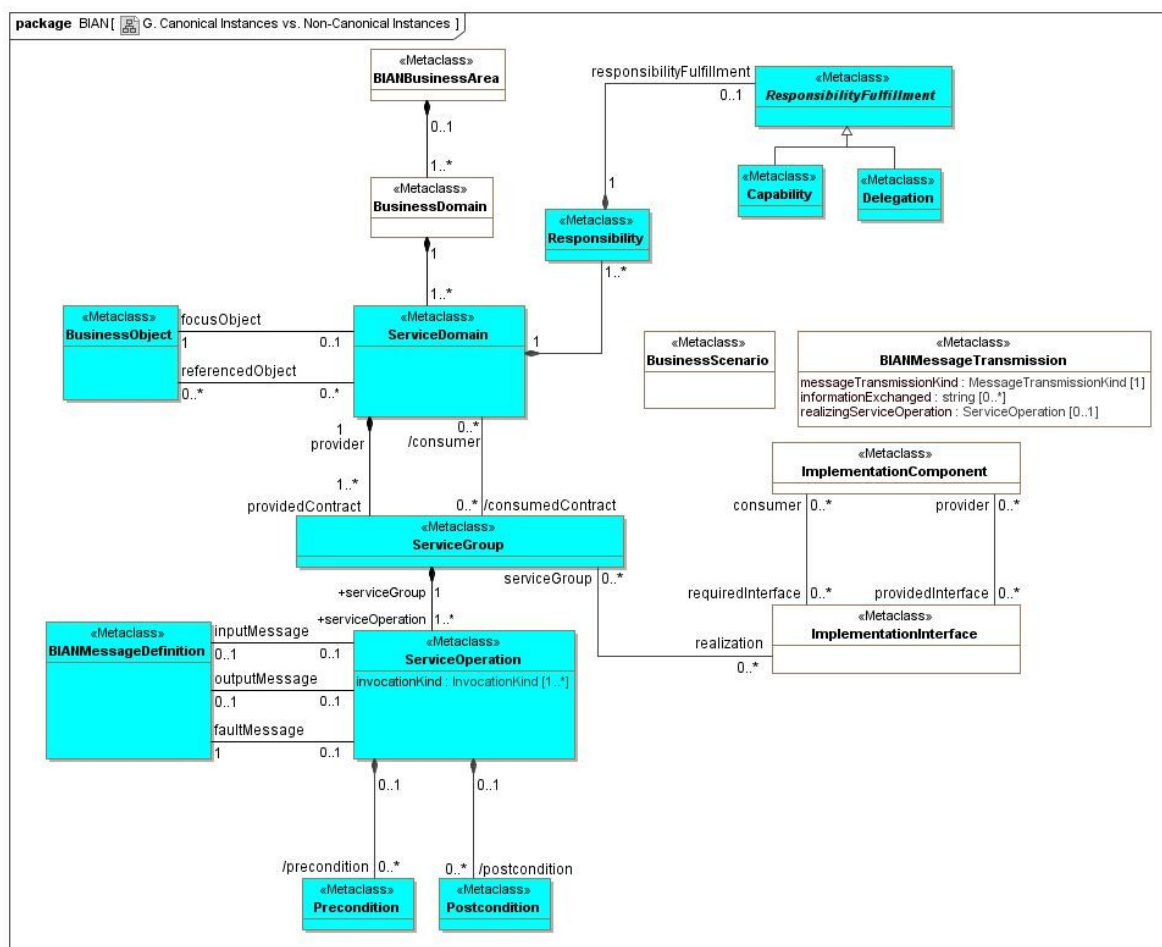


Figure 9. G. Canonical Instances vs. Non-Canonical Instances

This diagram provides a summary view of the BIAN Metamodel, using color coding to distinguish between elements of the metamodel whose instances are published as part of canonical BIAN standards (blue) and elements whose instances are not canonical (white).

See the explanations of the Service Landscape Viewpoint, Scenario Viewpoint, and Implementation Component Viewpoint, which describe the rationale behind the non-canonical nature of the instances of the metamodel elements coded in white on this diagram.

## Package – BIAN::Level1

The BIAN::Level1 package contains the elements of the core BIAN Metamodel that are relevant to Level 1 of BIAN's three-level architecture. Level 1 is the architecture's highest level of abstraction, where a ServiceDomain is defined in terms of its focus Business Object, its Responsibilities, and how it fulfills those Responsibilities.

The fact that an element is contained in this package does *not* mean it is irrelevant to Levels 2 and 3, but the fact that an element is not contained in this package *does* mean that it is irrelevant to Level 1.

## Additional Classes

The following classes are members of this package but have already been documented above:

[Class BIANBusinessArea](#), [Class BIANBusinessAssociationEnd](#), [Class BIANBusinessAttribute](#), [Class BIANConstraint](#), [Class BIANMessageTransmission](#), [Class BusinessDomain](#), [Class BusinessObject](#), [Class BusinessScenario](#), [Class Capability](#), [Class Delegation](#), [Class Notification](#), [Class RequestSatisfaction](#), [Class Responsibility](#), [Class ResponsibilityFulfillment](#), [Class ServiceDomain](#), [Class ServiceRole](#)

## Enumerations

[Enumeration MessageTransmissionKind](#)

## Package – BIAN::Level2

---

The BIAN::Level2 package contains metamodel elements that are relevant to Level 2 of BIAN's three-level architecture. Level 2 specifications add to Level 1 by defining a Service Domain's Service Groups, which are groups of Service Operations. The Service Operations are defined at a semantic level.

The fact that an element is contained in this package does *not* mean that it is irrelevant to Level 3, but *does* mean that it is not needed for Level 1 specifications.

## Additional Classes

The following classes are members of this package but have already been documented above:

[Class BIANChoiceComponent](#), [Class BIANMessageAssociationEnd](#), [Class BIANMessageAttribute](#), [Class BIANMessageBuildingBlock](#), [Class BIANMessageComponent](#), [Class BIANMessageDefinition](#), [Class Postcondition](#), [Class Precondition](#), [Class ServiceGroup](#), [Class ServiceOperation](#)

## Enumerations

[Enumeration InvocationKind](#)

## Package – BIAN::Level3

---

The BIAN::Level3 package contains metamodel elements that are relevant to Level 3 of BIAN's three-level architecture. Level 3 specifications add to Level 2 specifications by defining components that implement Service Groups and their ServiceOperations.

The fact that an element is contained in this package means that it is not needed for Level 1 and Level 2 specifications.

## Additional Classes

The following classes are members of this package but have already been documented above:

[Class ImplementationComponent](#), [Class ImplementationInterface](#)



## Package – BIAN\_ISO20022

---

This package contains diagrams that show the relationships between elements of the BIAN Metamodel and elements of the ISO 20022 Metamodel.

The BIAN Metamodel is a pure extension of the ISO20022 Metamodel that does not change the ISO20022 Metamodel. The extension is accomplished via subclassing relationships, whereby elements of the BIAN Metamodel subclass elements of the ISO20022 Metamodel. Since a subclass relationship (called a "Generalization" in UML parlance) is owned by the element playing the role of subclass, the subclassing relationships are contained within the BIAN::Level1 and BIAN::Level2 packages, and thus there are no metamodel elements that belong to this BIAN\_ISO20022 package; the package contains only diagrams. The metaclasses of the ISO20022 Metamodel that appear on those diagrams are listed without documentation; the documentation can be obtained from ISO.

### Comments

It is important to understand that ISO20022's RepositoryConcept element is at the top of the ISO20022 inheritance hierarchy, and thus defines properties that ISO20022 elements share in general. By making BIAN elements descendants of RepositoryConcept, BIAN elements take on those properties.

For example, BIAN elements have a name property by virtue of this inheritance, and thus BIAN elements don't contain their own name properties -- they simply use the name property that they inherit. RepositoryConcept contains many useful properties that BIAN elements need and would have to re-invent if not for their availability through this inheritance.

In general, extending the ISO20022 Metamodel provides BIAN elements with many needed properties required for a robust metamodel, and has the additional benefit of ensuring alignment with ISO20002.

There is an unfortunate name conflict whereby both ISO 20022 and BIAN have an element meant to represent a business area. We have to maintain both of these as distinct elements because we don't want to force the BIAN business areas, which can vary from bank to bank and are not canonical BIAN standards, to be the same as the ISO 20022 business areas, which have been officially defined by the governing ISO 20022 authorities. We have named the BIAN element BIANBusinessArea to distinguish it from the ISO 20022 BusinessArea element. In general where an element of the BIAN metamodel subclasses an ISO20022 element, such as BIANConstraint subclassing ISO20022 Constraint, we affix a "BIAN" prefix to the BIAN element's name, in order to make the distinction between the BIAN element and its ISO20022 superclass obvious. (An exception is BIAN's BusinessObject element, which subclasses ISO20022 BusinessComponent -- it already had a long-accepted name in BIAN that is different than the name of its ISO 20022 superclass.) However, the case of BIANBusinessArea is different, because it is not an extension of ISO20022 BusinessArea, but actually represents a different set of business areas.

## A. BIAN-ISO20022 Service Landscape Viewpoint

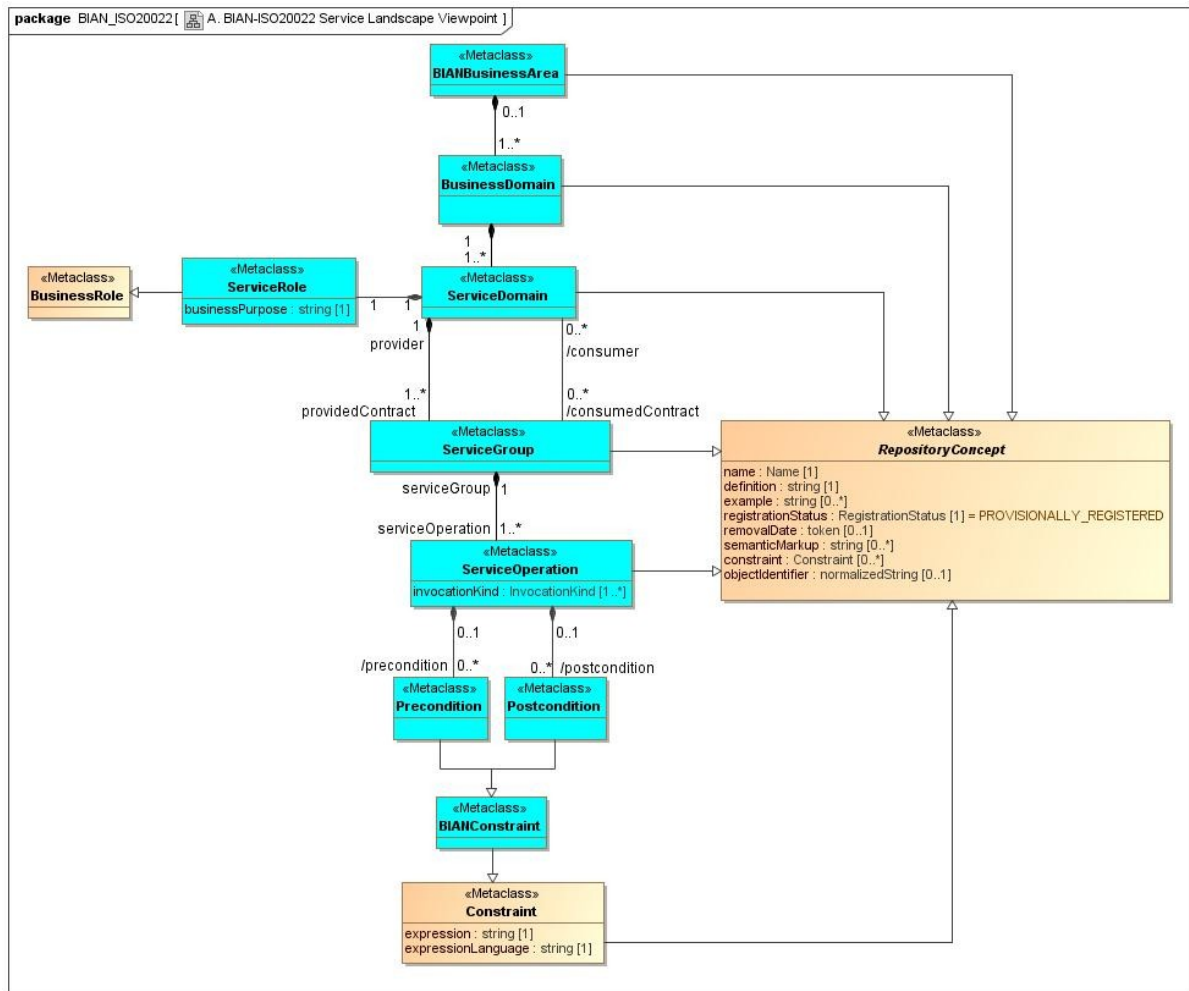


Figure 10. A. BIAN-ISO20022 Service Landscape Viewpoint

This diagram adds to the Service Landscape Viewpoint by showing the subclass relationships between the elements of the viewpoint and elements of the ISO20022 Metamodel.

Elements of the BIAN Metamodel are colored blue, while elements of the ISO20022 Metamodel are colored brown.

### Class BusinessRole

```
package ISO20022::ScopeLevel
```

### Class Constraint

```
package ISO20022::Metamodel
```

### Class RepositoryConcept

```
package ISO20022::Metamodel
```

## B. BIAN-ISO20022 Capabilities-Responsibilities-Delegation Viewpoint

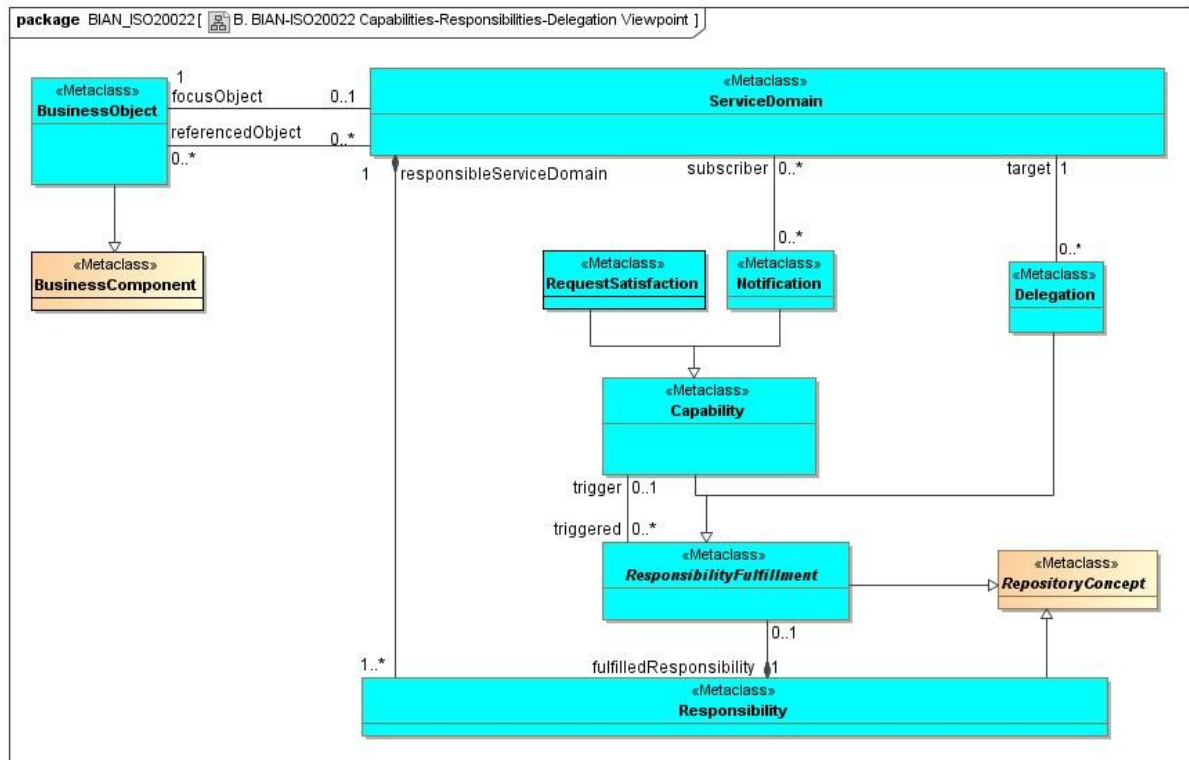


Figure 11. B. BIAN-ISO20022 Capabilities-Responsibilities-Delegation Viewpoint

This diagram adds to the Capabilities-Responsibilities-Delegation viewpoint by showing the subclass relationships between the elements of the viewpoint and elements of the ISO20022 Metamodel.

Elements of the BIAN Metamodel are colored blue, while elements of the ISO20022 Metamodel are colored brown.

### Class BusinessComponent

```
package ISO20022::ConceptualLevel::Static
```

## C. BIAN-ISO20022 Business Object Viewpoint

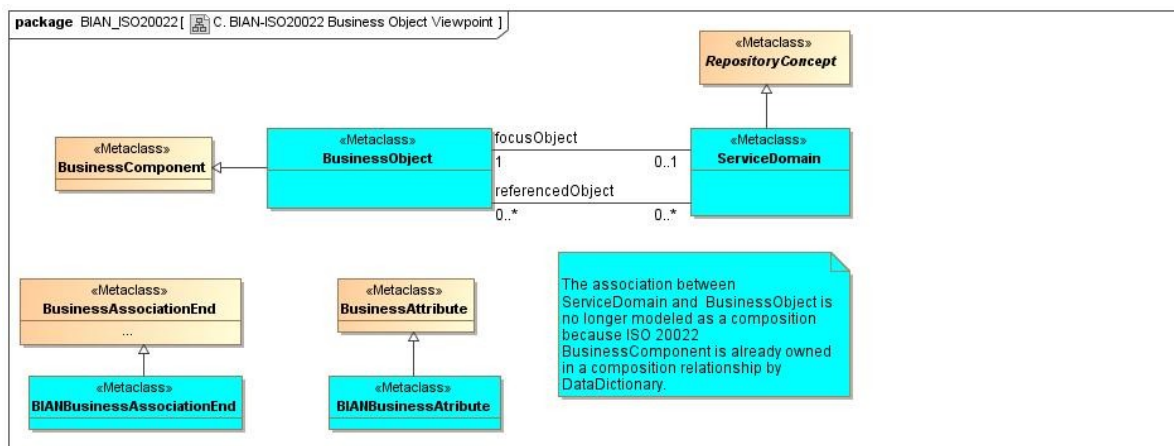


Figure 12. C. BIAN-ISO20022 Business Object Viewpoint

This diagram adds to the Business Object Viewpoint by showing the subclass relationships between the elements of the viewpoint and elements of the ISO20022 Metamodel.

Elements of the BIAN Metamodel are colored blue, while elements of the ISO20022 Metamodel are colored brown.

The choice of which ISO20022 elements should be extended rather than reused as-is is driven by analysis of which elements require the extra properties that the BIAN extensions inherit from `BIANElement` and `TaggableElement`.

### Class `BusinessAssociationEnd`

```
package ISO20022::ConceptualLevel::Static
```

### Class `BusinessAttribute`

```
package ISO20022::ConceptualLevel::Static
```

## D. BIAN-ISO20022 Message Viewpoint

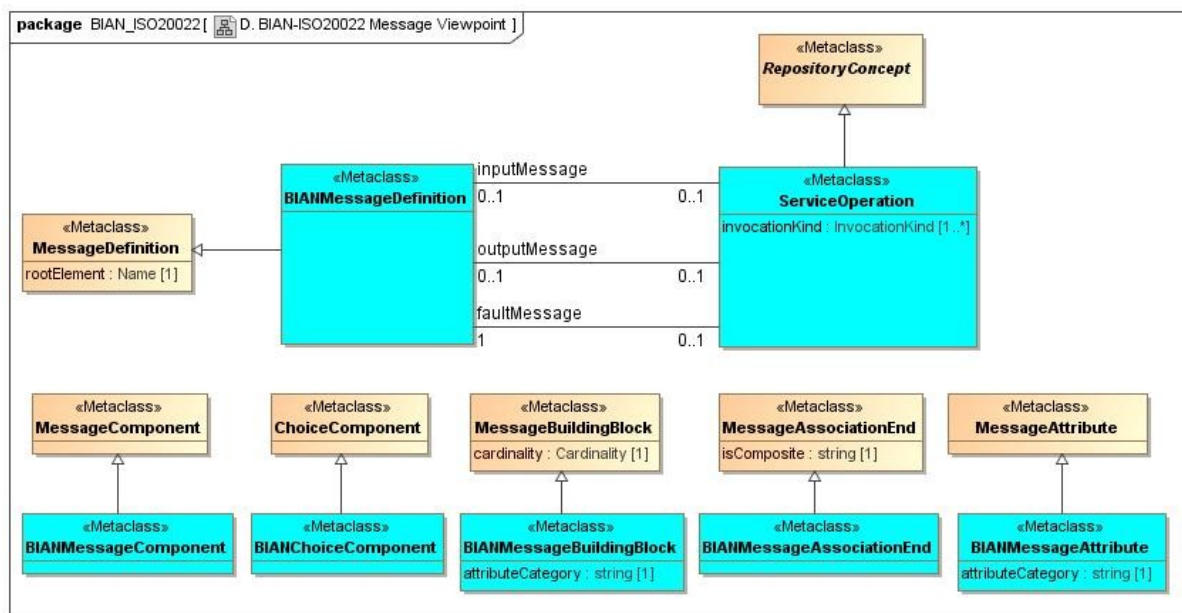


Figure 13. D. BIAN-ISO20022 Message Viewpoint

This diagram adds to the Message Viewpoint by showing the subclass relationships between the elements of the viewpoint and elements of the ISO20022 Metamodel.

Elements of the BIAN Metamodel are colored blue, while elements of the ISO20022 Metamodel are colored brown.

The choice of which ISO20022 elements should be extended rather than reused as-is is driven by analysis of which elements require the extra properties that the BIAN extensions inherit from `BIANElement` and `TaggableElement`.

### Links Between Elements of the Message and Business Object Viewpoints

The ISO20022 Metamodel supports connections between elements of the Business Object and Message viewpoints. The BIAN elements of these two viewpoints inherit the connections among their ISO20022 ancestors.

An ISO20022 `BusinessComponentTrace` is a link that supports two cases: establishing traceability from a `MessageComponent` to a `BusinessComponent`, and establishing traceability from a `MessageElement` to a `BusinessComponent`. The latter case supports scenarios in which a single `MessageElement` may be all that is required to capture the information from a `BusinessComponent` that needs to be transmitted in a message. For example, while a `BusinessComponent` may contain multiple `BusinessElements`, in some scenarios only one of the `BusinessElements` may need to be transmitted in a message, in which case creating a full-blown `MessageComponent` would be overkill.

An ISO20022 `BusinessElementTrace` establishes traceability from a `MessageElement` to a `BusinessElement`.

### Class ChoiceComponent

```
package ISO20022::LogicalLevel
```

### Class MessageAssociationEnd

```
package ISO20022::LogicalLevel
```

## Class MessageAttribute

package ISO20022::LogicalLevel

## Class MessageBuildingBlock

package ISO20022::LogicalLevel

## Class MessageComponent

package ISO20022::LogicalLevel

## Class MessageDefinition

package ISO20022::LogicalLevel

## E. BIAN-ISO20022 Scenario Viewpoint

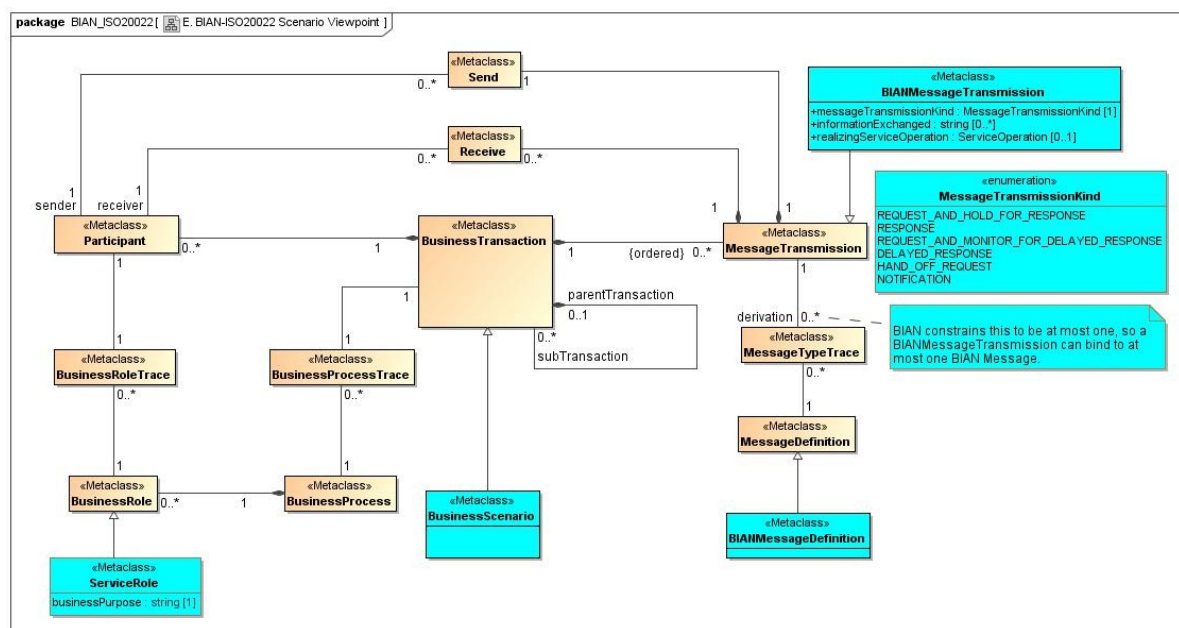


Figure 14. E. BIAN-ISO20022 Scenario Viewpoint

This diagram adds to the Scenario Viewpoint by showing the subclass relationships between the elements of the viewpoint and elements of the ISO20022 Metamodel. It makes visible the fact that a BIAN BusinessScenario is a fully conformant ISO 20022 BusinessTransaction.

Elements of the BIAN Metamodel are colored blue, while elements of the ISO20022 Metamodel are colored brown.

## Comments

As illustrated, in ISO 20022 a BusinessTransaction has a set of Participants and a set of MessageTransmissions. A Participant corresponds to a swimlane in a BIAN BusinessScenario diagram and to a Service Domain column in a BIAN BusinessScenario spreadsheet. A MessageTransmission corresponds to a line connecting two swimlanes in a BIAN BusinessScenario diagram and to a row in a BIAN BusinessScenario spreadsheet.

As further illustrated, in ISO 20022 the Send(ing) side of a MessageTransmission connects a Participant to the MessageTransmission, and the Receive(ing) side connects the other Participant to the MessageTransmission.

In ISO 20022 a MessageDefinition can be bound to a MessageTransmission via MessageTransmission's derivation property. Analogously, in BIAN a BIANMessageDefinition can be bound to a BIANMessageTransmission. Such binding connects a message content model to the mere act of transmission. The separation of concerns between the model of the transmission and the model of the message content makes it possible to model BusinessTransactions for which message content models have not yet been defined.

### Class BusinessProcess

```
package ISO20022::ScopeLevel
```

### Class BusinessProcessTrace

```
package ISO20022::ScopeToConceptualTransformation
```

### Class BusinessRoleTrace

```
package ISO20022::ScopeToConceptualTransformation
```

### Class BusinessTransaction

```
package ISO20022::ConceptualLevel::Dynamic
```

### Class MessageTransmission

```
package ISO20022::ConceptualLevel::Dynamic
```

### Class MessageTypeTrace

```
package ISO20022::ConceptualToLogicalTransformation
```

### Class Participant

```
package ISO20022::ConceptualLevel::Dynamic
```

### Class Receive

package ISO20022::ConceptualLevel::Dynamic

### Class Send

package ISO20022::ConceptualLevel::Dynamic

## Enumerations in this Package Documented Above

[Enumeration MessageTransmissionKind](#)

## Package – BIAN\_SemanticMetadata

---

The BIAN\_SemanticMetadata package contains a diagram that shows the relationships between the elements of the BIAN package and elements of the SemanticMetadata package.

The binding is achieved via subclass relationships whereby elements of the BIAN package subclass the TaggableElement element of the SemanticMetadata package.

Since a subclass relationship (called a "Generalization" in UML parlance) is owned by the element playing the role of subclass, the subclass relationships are contained within the BIAN::Level1, BIAN::Level2, and BIAN::Level3 packages, and thus there are no metamodel elements in this BIAN\_SemanticMetadata package; the package contains only a diagram that shows the subclass relationships.

### Comment

The SemanticMetadata package is potentially applicable to other standards such as XBRL and ISO20022.



## BIAN - Semantic Metadata Binding

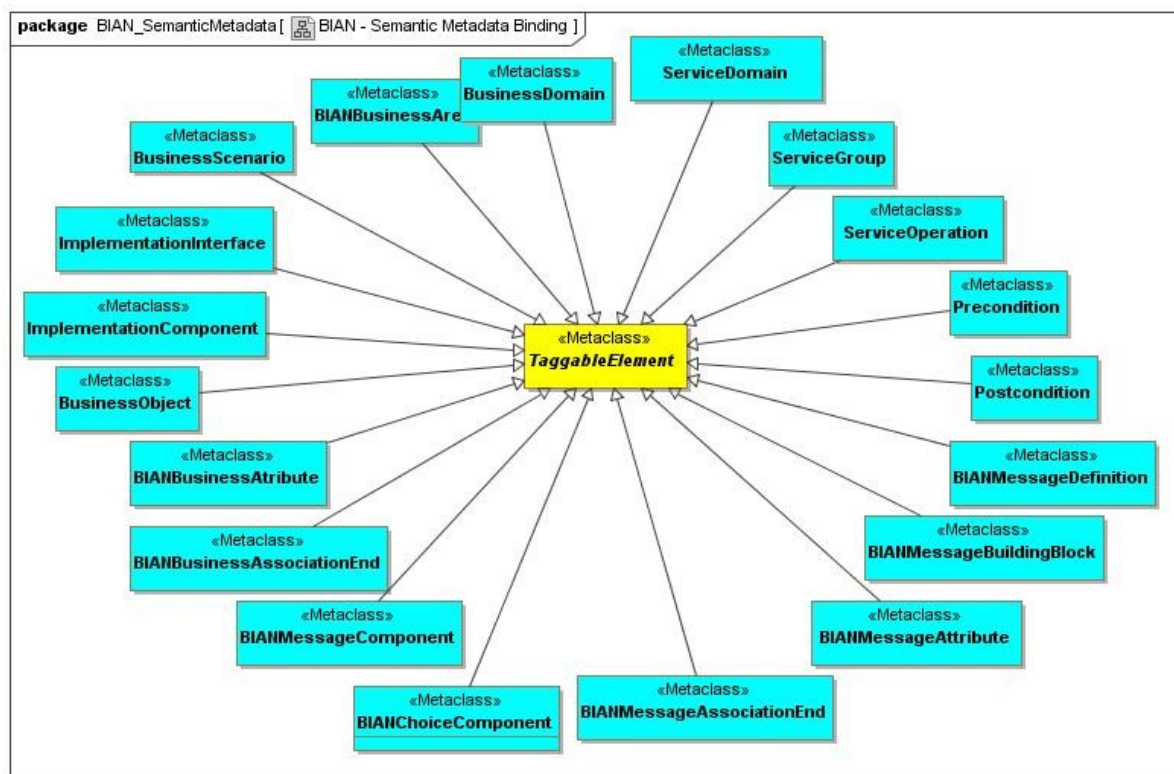


Figure 15. BIAN - Semantic Metadata Binding

There are a number of elements of the BIAN Metamodel in whose instances we wish to embed SemanticMetadataTags. These metamodel elements directly subclass the SemanticVocabulary package's TaggableElement. Note that we use multiple inheritance, since the taggable BIAN elements are also descendants of BIANElement and of ISO 20022's RepositoryConcept.

### Class TaggableElement

```
package SemanticMetadata
```

A TaggableElement is an element in which a SemanticMetadataTag can be embedded.

All elements in the BIAN metamodel in whose instances such tags can be embedded are descendants (in the subclassing sense) of TaggableElement.

#### Direct Superclasses

#### Associations



```
embeddedTag : EmbeddedTag [0..*]
```

The EmbeddedTags that are embedded in the TaggableElement. The association is a composition in which TaggableElement plays the role of composite and EmbeddedTag plays the role of component.

## Attributes

## Constraints

# Package – ISO20022\_SemanticMetadata

---

Defines the relationships between the ISO 20022 metamodel and the SemanticVocabulary metamodel.

Note that the metamodeling standards and tools have limitations that don't permit the two metamodels to be entirely orthogonal and then combined -- i.e., the techniques at hand don't support the kind of loose coupling that would be preferable.

For example, it would be ideal to be able to loosely couple the SemanticVocabulary metamodel with the ISO 20022 metamodel and with the XBRL metamodel. So when we bind the SemanticVocabulary metamodel with the ISO 20022 metamodel, we would like SemanticConcept to be a subtype of RepositoryConcept, whereas when we bind the SemanticVocabulary metamodel with the XBRL metamodel we would like SemanticConcept to be a subclass of something in the XBRL metamodel that serves roughly the same purpose. This would require a loose subtyping construct that could be dynamically applied and unapplied. OMG is working on updates to the MDA stack that will support this, but that is still a work in progress.

As a result, we are forced into a static binding of the two metamodels, so the fact that SemanticConcept is a subtype of ISO 20022's RepositoryConcept has to be baked into SemanticConcept. That means that we can't use the same SemanticConcept element when binding to XBRL. The best we can do is at least separate the ISO 20022 and SemanticVocabulary concepts in the diagrams; each diagram is a view of a subset of the full metamodel. So in the diagrams associated with the SemanticVocabulary package, we don't show the relationships to the ISO 20022 elements, putting those in a separate diagram associated with this package named ISO20022SemanticVocabulary.

## Binding to RepositoryConcept

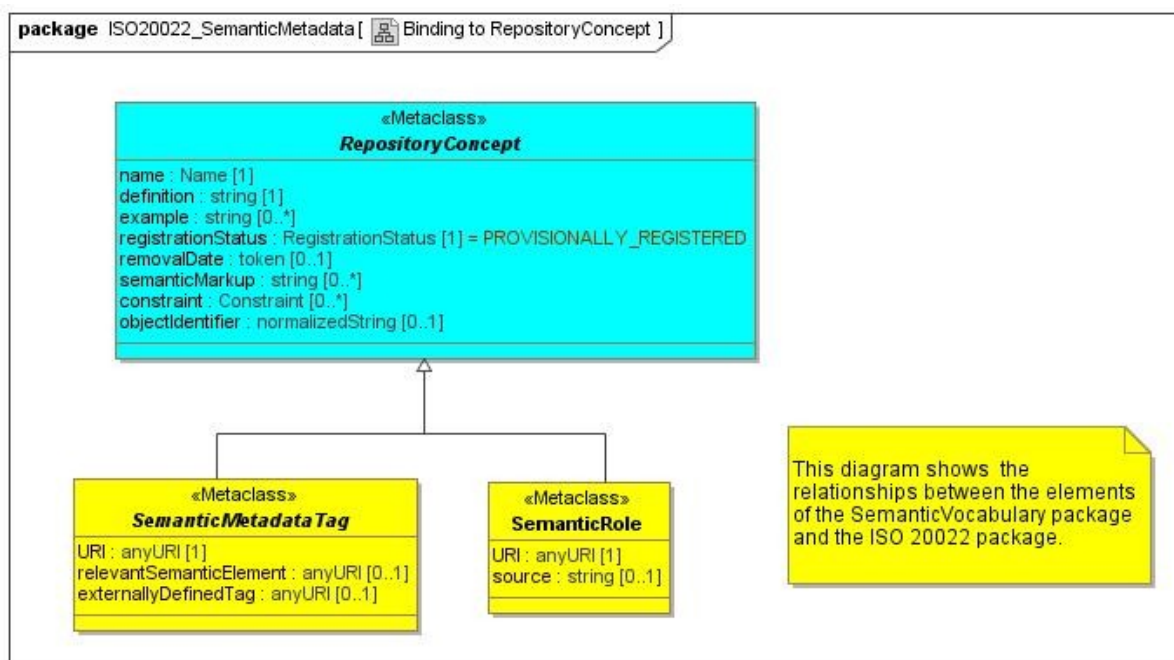


Figure 16. Binding to RepositoryConcept

### Class SemanticMetadataTag

package SemanticMetadata

A SemanticMetadataTag declares that a specific semantic element, which lives in some controlled vocabulary, plays a specific SemanticRole in the element being tagged.

In the typical case, the tag points to the semantic element in the controlled vocabulary via the tag's relevantSemanticElement property, whose value is a URI. Sometimes, however, the tag merely references a tag that is defined elsewhere, via the SemanticMetadataTag's externallyDefinedTag property, whose values are also URIs. As background to the latter case, consider that UN/CEFACT is in the process of defining a means to construct business context tags that are quite complex directed acyclic graphs; however, the complex tags always have URIs as identifiers, so, rather than reproduce a metamodel of corresponding complexity, we have a means to point to tags of arbitrary complexity via URIs.

#### Direct Superclasses

[RepositoryConcept](#)

#### Associations

semanticRole : [SemanticRole](#) [1]

The SemanticRole that the relevantSemanticElement (or the semantic element referenced by the externallyDefinedTag) plays in the element being tagged.

#### Attributes

URI : anyURI [1]

The URI that identifies the tag.

relevantSemanticElement : anyURI [0..1]

The URI of the semantic element that is relevant to the element being tagged. Typically the relevant semantic element lives in a controlled vocabulary or ontology.

externallyDefinedTag : anyURI [0..1]

The URI of a tag that lives in some external location. When a value is assigned to this property, it means that, rather than point to the relevant semantic element directly, the tag references another tag that points to the relevant semantic element.

### Constraints

Constraint relevantSemanticElementXORexternallyDefinedTag:

Exactly one of the following two properties of SemanticMetadataTag must be non-empty: relevantSemanticElement and externallyDefinedTag.

In other words: The tag either points to a relevant semantic element, such as an entry in a controlled vocabulary or a provision in a government regulation, or it points to an externally defined tag that points to the relevant elements.

```
[relevantSemanticElement->isEmpty() xor externallyDefinedTag->isEmpty()]
```

## Class SemanticRole

```
package SemanticMetadata
```

A SemanticRole is a role that an element of a controlled vocabulary or ontology can play in an element of a service definition.

Elements of service definitions include BusinessAreas, BusinessDomains, ServiceDomains, ServiceOperations, Messages, BusinessObjects, and so on.

### [Direct Superclasses](#) [RepositoryConcept](#)

### Associations



```
semanticMetadataTag : SemanticMetadataTag [0..*]
```

The SemanticMetadataTags for which this is the SemanticRole.

### Attributes

URI : anyURI [1]

The URI that identifies the SemanticRole.

source : string [0..1]

Documentation of the source of the SemanticRole, i.e. where the role is defined.

For example, the source of the object class Semantic Role is ISO 11179.

## Constraints

# Package – SemanticMetadata

The SemanticMetadata package shows all the metamodel elements that support the use of business vocabularies. The requirement for BIAN that it addresses is the ability to

- maintain a business vocabulary and
- to embed metadata in the elements of our service definitions that ties those elements to the entries in the vocabulary

As BIAN looked at what is going on with semantic technology in industry, particularly technology that supports the maintenance of controlled vocabularies, we determined that there was no point in duplicating work being done elsewhere that we can leverage.

Thus, this part of the metamodel assumes that controlled vocabularies are out there in the world and accessible via URIs. There are free wiki tools available that allow any community to establish and maintain a controlled vocabulary, each entry of which is accessible by a URI (in this case the URI for a vocabulary entry is a URL).

So when we remove the modeling of controlled vocabulary from this part of the metamodel, what is left? It is just the metamodel of the metadata tags that we embed in the elements of our service definitions, i.e. the tags that point to controlled vocabulary entries via their URIs.

## Semantic Metadata

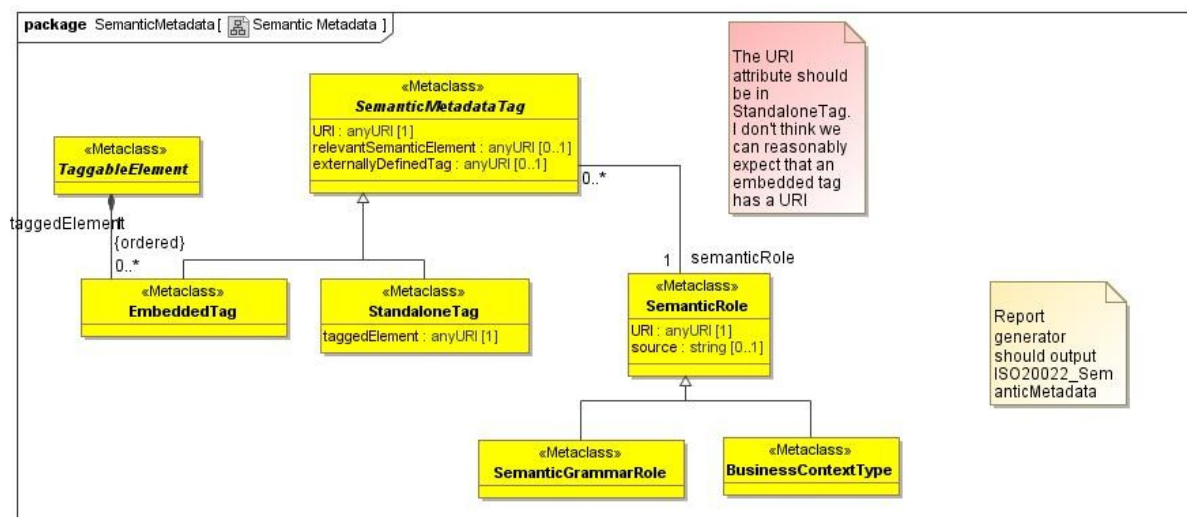


Figure 17. Semantic Metadata

This diagram shows the metamodel elements that support the use of business vocabularies.

The key element is SemanticMetadataTag, which represents a tag that can either be embedded in a TaggableElement (as is the case for an EmbeddedTag) or can standalone by pointing to the tagged element via a URI (as is the case for a StandaloneTag).

A SemanticMetadataTag declares that, for the taggedElement, a semantic concept defined in a controlled vocabulary plays a specific SemanticRole.

### Class BusinessContextType

package SemanticMetadata

A BusinessContextType is a SemanticRole that categorizes the kinds business context values with which elements can be tagged.

For example, UN/CEFACT's Core Components standard (CCTS) has defined the following types of business context values, each of which we can define as an instance of BusinessContextType:

- Business process: The business process(es) with which the element is associated
- Business process role: The role(s) directly involved in the business process(es) that are relevant to the element, e.g. shipper
- Supporting role: The role(s) indirectly involved in the business process; e.g., a data element in an order response from seller to buyer could be required by a third-party shipper
- Geopolitical scope: The continent, economic region, country, or region for which the element is relevant
- Industry classification: The industry or industries for which the element is relevant
- Product classification: The product class(es) for which the element is relevant
- Official constraint: A legal requirement(s) associated with the element; e.g., an element could be required by Sarbanes-Oxley
- System capability: Indicates that the element is there in order to address the limitations of a computing system(s)

*Direct Superclasses*  
[SemanticRole](#)

*Associations*

*Attributes*

*Constraints*

### Class EmbeddedTag

package SemanticMetadata

A SemanticMetadataTag that is embedded in an element of a BIAN service definition.

Elements of service definitions in which SemanticMetadataTags can be embedded include BusinessAreas, BusinessDomains, ServiceDomains, ServiceOperations, Messages, BusinessObjects, and so on.

*Direct Superclasses*  
[SemanticMetadataTag](#)

*Associations*



taggedElement : [TaggableElement](#) [1]

The element in which the EmbeddedTag is embedded. The association is a composition in which TaggableElement plays the role of composite and EmbeddedTag plays the role of component.

### Attributes

### Constraints

## Class SemanticGrammarRole

```
package SemanticMetadata
```

A SemanticRole that is part of a semantic grammar.

For example, the semantic grammar defined by BIAN for Service Operations is:

<action><object class>[<property>]

In a ServiceOperation named Update\_Portfolio\_NetPresentValue, the action is Update, the object class is Portfolio, and the property is NetPresentValue; action, object, and property are SemanticRoles that are composed to form the semantic grammar for a ServiceOperation.

A semantic grammar can be used to drive a naming convention, but that is a secondary purpose for such a grammar. The primary purpose is to be able to define metadata tags that declare the SemanticRoles that elements of a controlled vocabulary or ontology play in the element being tagged.

In the example, we could construct three SemanticMetadataTags based on these three roles, one that declares that Update plays the action role, another that declares that Portfolio plays the object class role, and another that declares that Net Present Value plays the property role.

For comparison sake, the semantic grammar that BIAN uses for an element of a business object is:

<object class><property>

This particular semantic grammar is actually defined in ISO 11179. Here we see that the object class SemanticRole is used in more than one semantic grammar.

See the SemanticMetadata::Examples package for detailed examples of the use of SemanticGrammarRoles.

### [Direct Superclasses](#) [SemanticRole](#)

### Associations

### Attributes

### Constraints

### Class StandaloneTag

package SemanticMetadata

A SemanticMetadataTag that points to the tagged element via a URI rather than being embedded in the tagged element.

It is unlikely that BIAN will define StandaloneTags in the near to medium term, because the BIAN Metamodel provides slots for embedding tags in elements of service definitions. We define this element nonetheless, because we wish to promote the use of semantic metadata in other venues where standalone tags may be essential. Standalone tags are necessary when there is no physical slot to hold the tag in the element being tagged, which is the case when elements of legacy systems need to be tagged.

[\*Direct Superclasses\*](#)  
[SemanticMetadataTag](#)

#### *Associations*

#### *Attributes*

taggedElement : anyURI [1]

The URI of the element being tagged.

#### *Constraints*

## SemanticMetadata::Examples

---

The SemanticMetadata::Examples package has a set of examples of SemanticMetadataTags embedded in elements of BIAN service definitions. The examples are UML instance diagrams that show the instances of the metamodel elements that are created in order to embed several kinds of tags in BIAN metamodel elements.

The boxes represent instances of the metamodel's metaclasses. The lines represent instances of the metamodel's associations (in UML, an instance of an association is called a "link"). Note that the association end names from the metamodel are shown at the ends of the links.



## A. Example - Tags Embedded in the Definition of a Service Operation

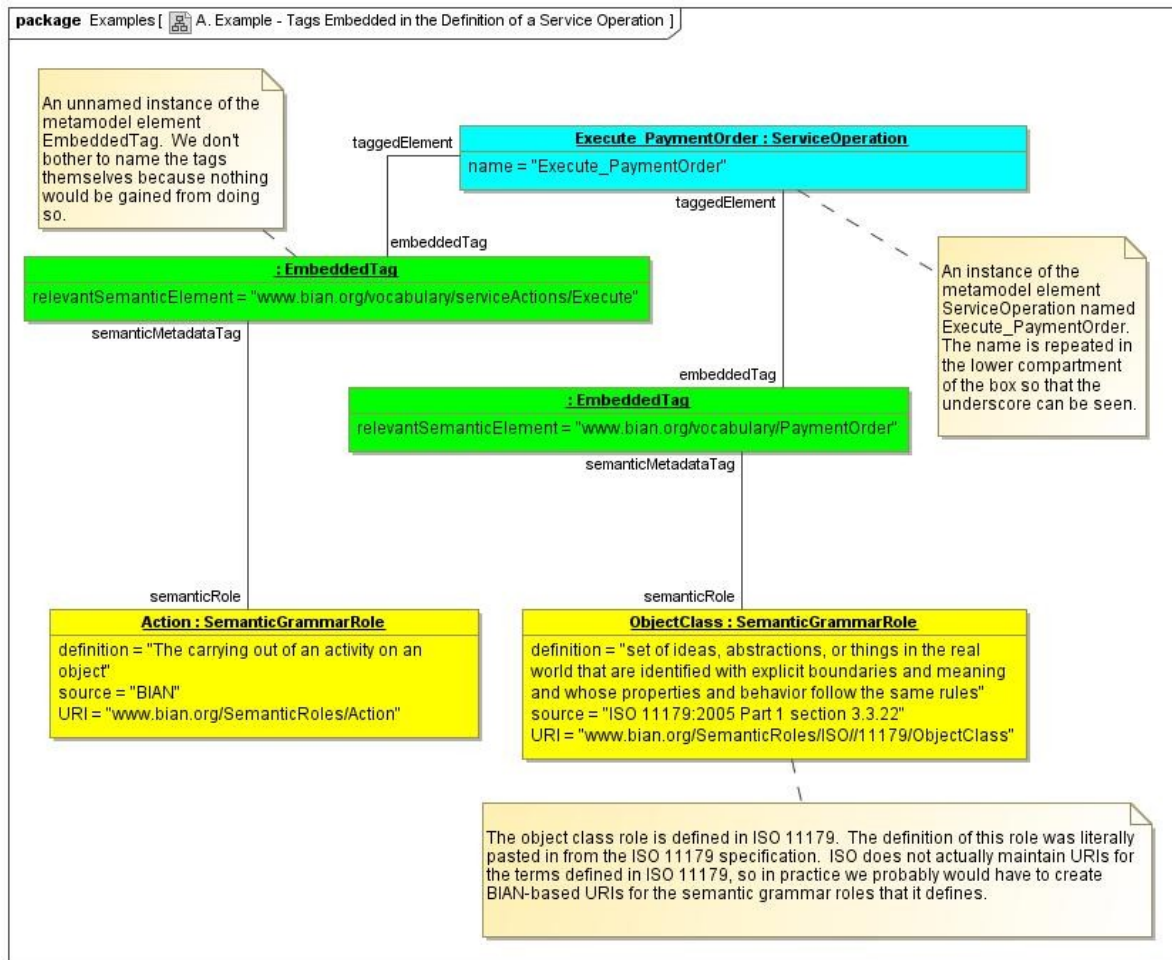


Figure 18. A. Example - Tags Embedded in the Definition of a Service Operation

This example illustrates the embedding of three SemanticMetadataTags in the definition of a ServiceOperation:

- The first tag (reading the EmbeddedTags from right to left) says that the Execute concept plays the Action role in the ServiceOperation, and points to the definition of Execute in a controlled vocabulary.
- The second tag says that the Payment Order concept plays the object class role, and points to the definition of Payment Order.

Note that the metamodel makes it possible to embed tags that point to definitions defined outside of BIAN, as long as there is a URI that can serve as the pointer.

## B. Example - Additional Semantic Role: Property

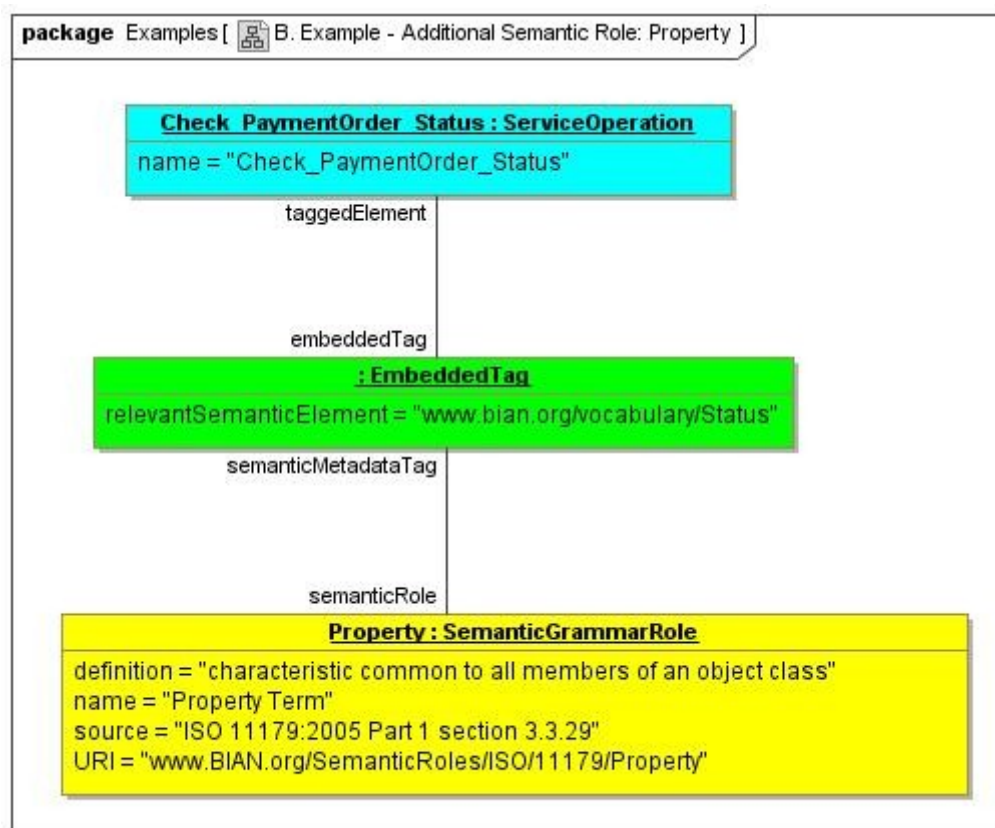


Figure 19. B. Example - Additional Semantic Role: Property

This example introduces another SemanticGrammarRole defined by ISO 11179: property. A ServiceOperation always has an action concept and an object class concept, and sometimes has a property concept. In this operation, the Status concept plays the property role.

## C. Example - Additional Semantic Role: Object Class Qualifier

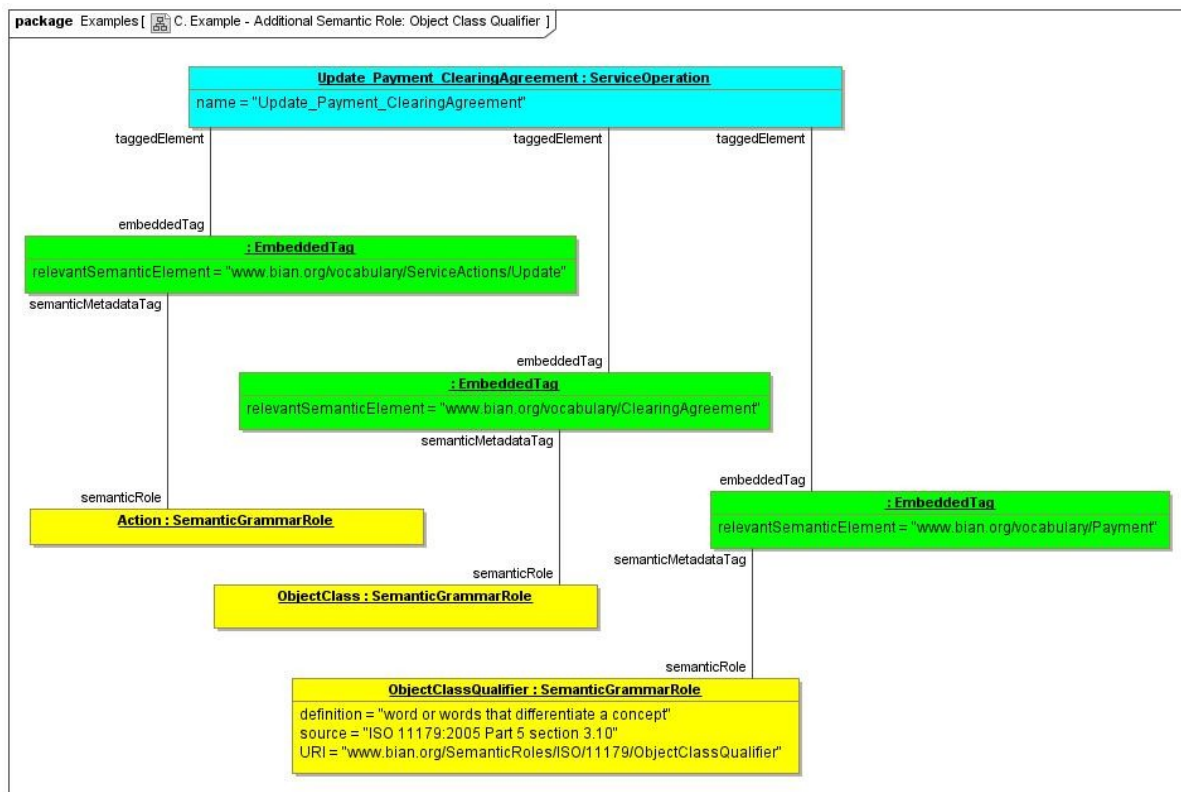


Figure 20. C. Example - Additional Semantic Role: Object Class Qualifier

This example highlights an additional **SemanticGrammarRole** that is sometimes used: object class qualifier. In the **ServiceOperation** named **Update\_Payment\_ClearingAgreement**, **Clearing Agreement** concept plays the object class role, and **Payment** plays the object class qualifier role. Qualifier roles are essentially secondary roles that can be applied with respect to not only the object class role, but also with respect to the property and action roles.

Multiple qualifiers can be strung together. For example, we could envision an operation **Update\_Primary\_Payment\_ClearingAgreement**, where **Payment** and **Primary** both play the object class qualifier role; in such a case, the order of the tags is significant in that **Payment** qualifies **ClearingAgreement** and **Primary** qualifies **Payment\_ClearingAgreement**; thus the metamodel preserves the order of the tags -- see the association between **TaggableElement** and **EmbeddedTag**, which is an ordered association.

## D. Example - Tags based on Compound Vocabulary Entries

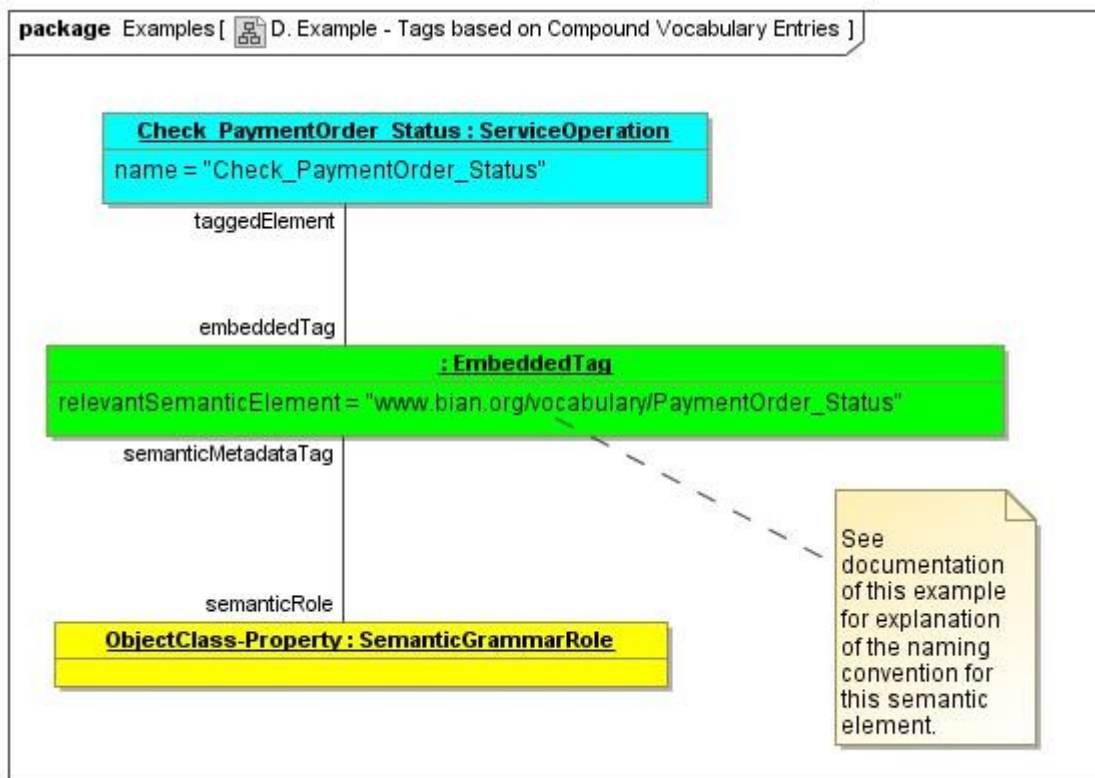


Figure 21. D. Example - Tags based on Compound Vocabulary Entries

This example introduces an additional SemanticGrammarRole that supports domain experts who assess that the definition of Payment Order Status is more than the sum of the definition of Payment Order and the definition of Status.

The relevantSemanticElement is an entry in the BIAN controlled vocabulary that defines Payment Order Status. Since Payment Order Status is a combination of an object class concept (Sales Order) and a property concept (Acknowledgment), we define a SemanticGrammarRole named "ObjectClass-Property" that is tailored for concepts that are object class - property combinations. In effect, this SemanticGrammarRole is a compound role, combining the object class and property roles. This compound role is applied to object class - property combinations.

The tag, embedded in the Check\_PaymentOrder\_Status service operation, declares that the Sales Order Acknowledgment concept plays the Semantic Grammar Role of object class-property in the Service Operation.

Note that for vocabulary entries that are combinations of concepts, such as a combination of an object class concept and a property concept in this example, we use the same convention for naming the compound entry that we use for naming the elements in our service definitions: Terms that represent concepts in the controlled vocabulary are capitalized and separated from each other using underscores; a term composed of two words is represented by combining the words via camel case. For example, Payment Order is a two-word term for a single concept, so it is represented as "PaymentOrder" while Status is a term for a separate concept and thus "Status" is separated from "PaymentOrder" by an underscore. The Check\_PaymentOrder\_Status operation follows the same naming convention.

## E. Example - Illustration that Semantic Roles are Defined Only Once

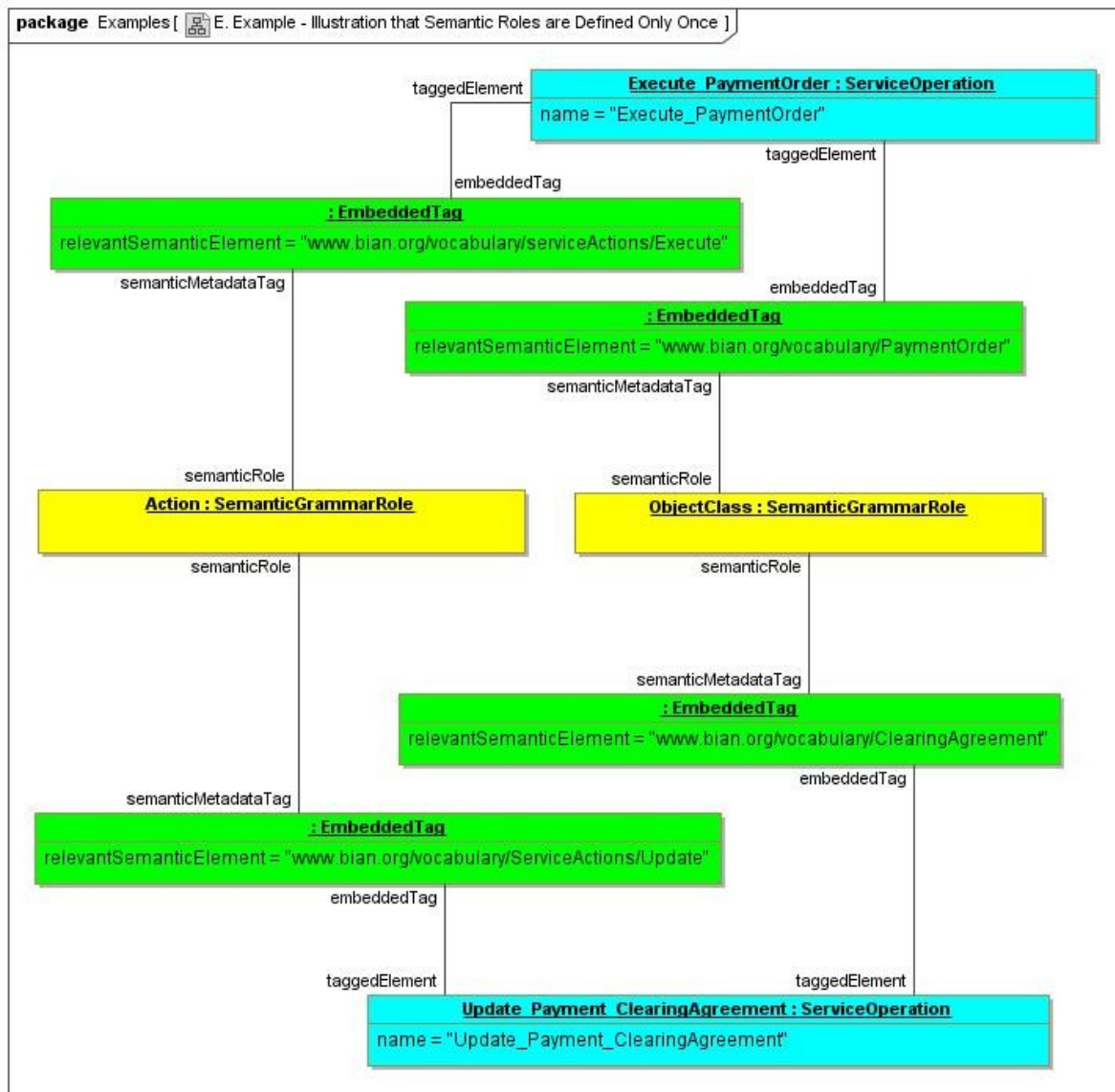


Figure 22. E. Example - Illustration that Semantic Roles are Defined Only Once

This example illustrates the fact that SemanticRoles are defined once conceptually. The "Action" and "ObjectClass" instances of SemanticGrammarRole referenced by the tags embedded in the Execute\_PaymentOrder ServiceOperation are also referenced by the tags embedded in the Update\_Payment\_ClearingAgreement Service Operation. (The ObjectClass instance of SemanticGrammarRole could also be referenced by tags embedded in the attributes of business objects and message components, although that is not illustrated here.)



## F. Example - Business Context Tags

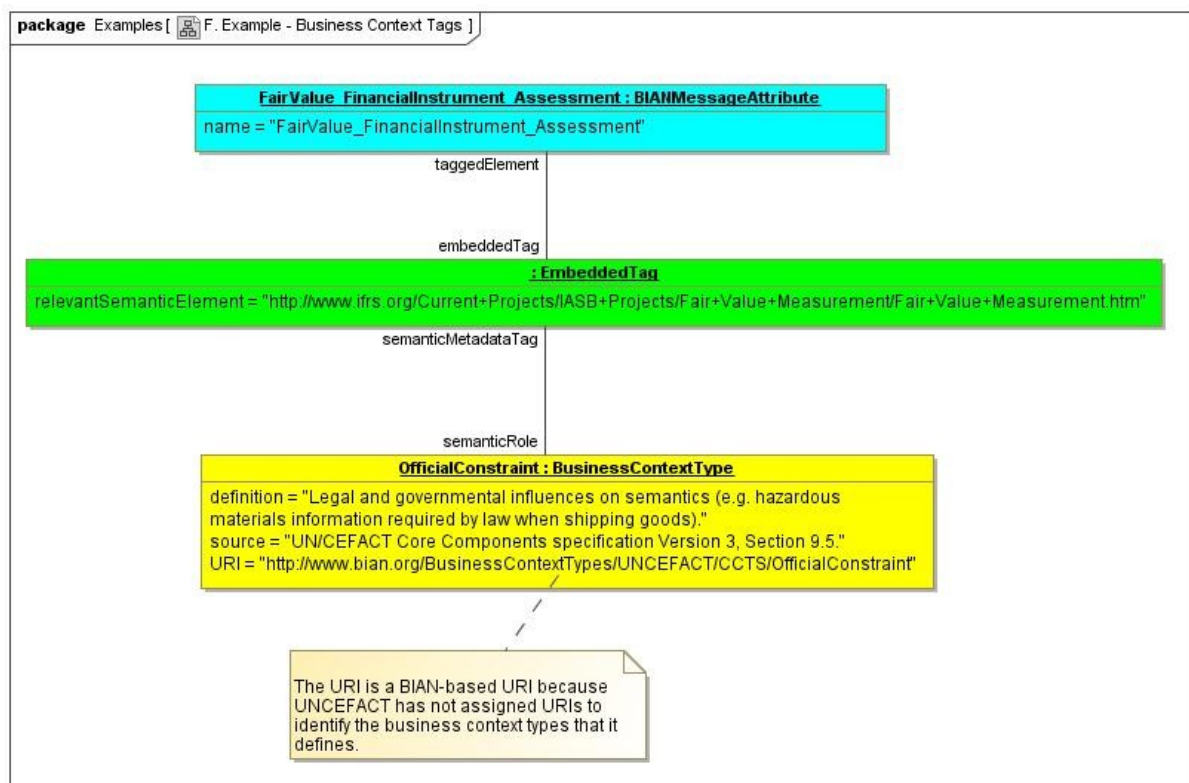


Figure 23. F. Example - Business Context Tags

This example illustrates a business context tag embedded in an attribute of a message component. The tag declares that the fair value provision of the International Accounting Standards Board (IASB) is relevant to the business context of this message attribute. The BusinessContextType for the tag is official constraint, which is a type of business context defined in the UN/CEFACT Core Components specification.

## G. Example - Referencing Externally Defined Business Context Tags

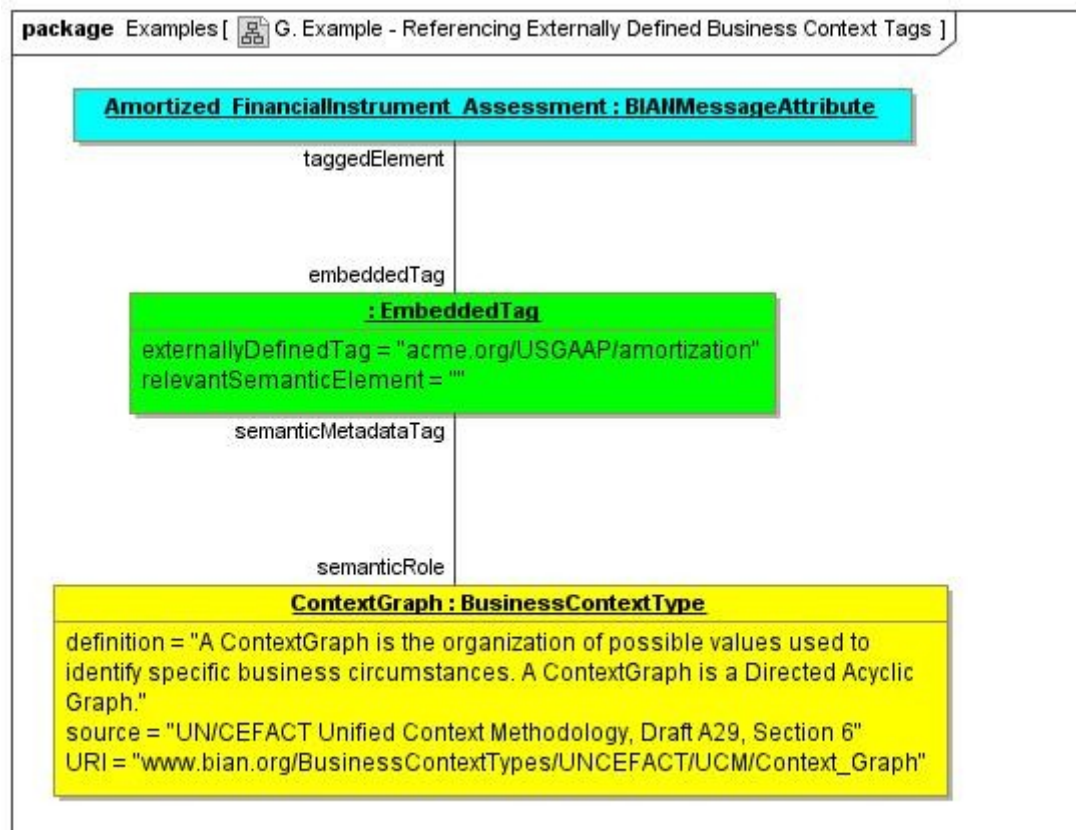


Figure 24. G. Example - Referencing Externally Defined Business Context Tags

This example illustrates the case where the tag does not directly reference the `relevantSemanticElement`, but, instead, references an `externallyDefinedTag`.

There are standards bodies that are working on models of business context that can be quite complex. For example, UN/CEFACT is working on business context descriptions that can be arbitrarily complex directed acyclic graphs. BIAN strives to co-exist with and leverage such complex models, rather than replicate their architecture in the metamodel. In the example, the tag simply points to an externally defined UN/CEFACT-style context graph. This approach is possible because the relevant UN/CEFACT specification (which is still a draft) specifies that a context graph has to have a URI.

## H. Example - Functional Pattern Tags

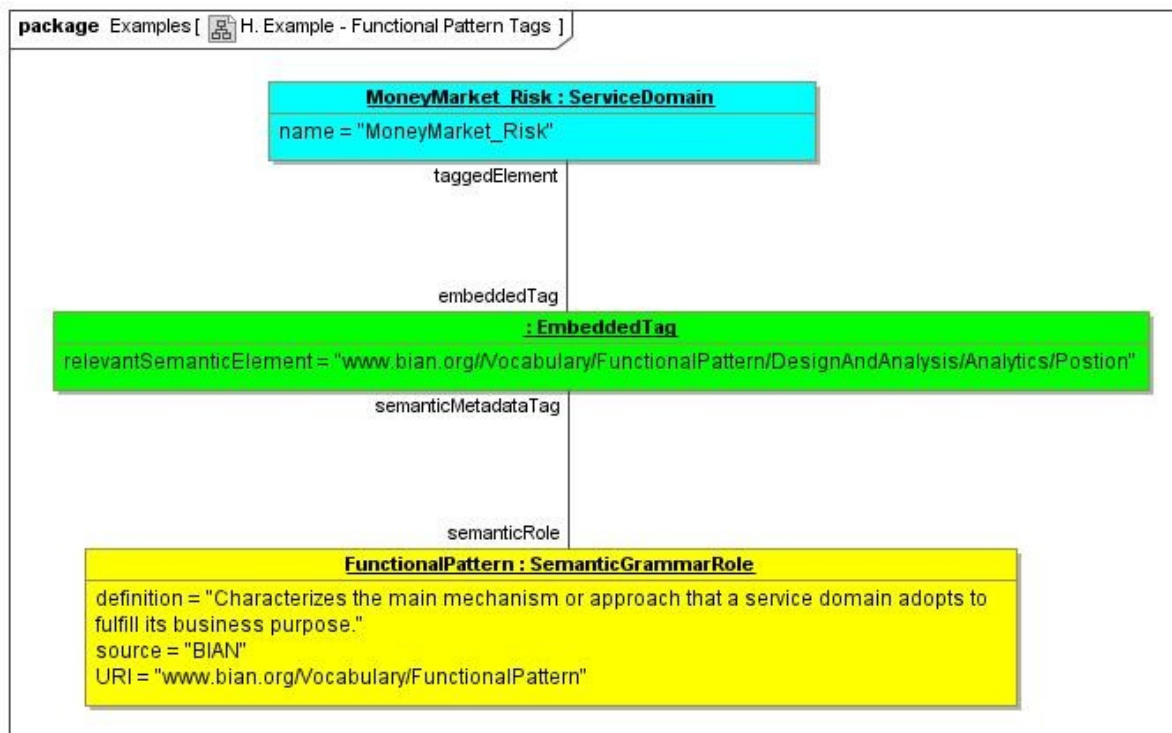


Figure 25. H. Example - Functional Pattern Tags

This example illustrates how we can embed additional semantic metadata tags in elements of BIAN service definitions to avoid hardwiring into the BIAN metamodel the notion of functional patterns.

In the course of updating the BIAN Service Landscape, a hierarchical set of functional patterns was proposed as metadata that categorize ServiceDomains. However, we lack experience with using functional patterns as metadata for this purpose. Thus, rather than hardwiring a functional pattern attribute into the ServiceDomain class, in this example we use the generic semantic metadata tagging facility to embed functional pattern metadata in a ServiceDomain definition. The same technique can be used for other kinds of metadata whose type we would rather not hardwire into the metamodel, such as business object categories.



# Appendix A: Tree

---

## Class Hierarchy

[BIAN::BIANElement](#)

[BIAN::Level1::BIANBusinessArea](#)

[BIAN::Level1::Responsibility](#)

[BIAN::Level1::ResponsibilityFulfillment](#)

[BIAN::Level1::Capability](#)

[BIAN::Level1::RequestSatisfaction](#)

[BIAN::Level1::Notification](#)

[BIAN::Level1::Delegation](#)

[BIAN::Level1::BusinessDomain](#)

[BIAN::Level1::ServiceDomain](#)

[BIAN::Level1::BusinessScenario](#)

[BIAN::Level1::BusinessObject](#)

[BIAN::Level1::BIANBusinessAttribute](#)

[BIAN::Level1::BIANBusinessAssociationEnd](#)

[BIAN::Level1::BIANConstraint](#)

[BIAN::Level2::Precondition](#)

[BIAN::Level2::Postcondition](#)

[BIAN::Level1::BIANMessageTransmission](#)

[BIAN::Level1::ServiceRole](#)

[BIAN::Level2::BIANMessageDefinition](#)

[BIAN::Level2::ServiceOperation](#)

[BIAN::Level2::ServiceGroup](#)

[BIAN::Level2::BIANMessageComponent](#)

[BIAN::Level2::BIANMessageBuildingBlock](#)

[BIAN::Level2::BIANMessageAttribute](#)

[BIAN::Level2::BIANMessageAssociationEnd](#)

[BIAN::Level2::BIANChoiceComponent](#)

[BIAN::Level3::ImplementationInterface](#)

[BIAN::Level3::ImplementationComponent](#)

[ISO20022::ConceptualLevel::Dynamic::Receive](#)

[BIAN::RelatedExternalElement](#)

[ISO20022::RepositoryConcept](#)

[ISO20022::ConceptualLevel::Static::BusinessAttribute](#)

[BIAN::Level1::BIANBusinessAttribute](#)

[ISO20022::ConceptualLevel::Static::BusinessElement](#)

[ISO20022::ConceptualLevel::Static::BusinessAssociationEnd](#)

[BIAN::Level1::BIANBusinessAssociationEnd](#)

[ISO20022::ConceptualLevel::Static::BusinessAttribute](#)

[BIAN::Level1::BIANBusinessAttribute](#)

[ISO20022::ConceptualLevel::Dynamic::Participant](#)

[ISO20022::ConceptualLevel::Dynamic::MessageTransmission](#)

[BIAN::Level1::BIANMessageTransmission](#)

[ISO20022::TopLevelDictionaryEntry](#)

[ISO20022::DataTypes::DataType](#)

[ISO20022::DataTypes::Binary](#)

[ISO20022::DataTypes::YearMonth](#)

[ISO20022::DataTypes::String](#)

[ISO20022::DataTypes::Text](#)

[ISO20022::IdentifierSet](#)

[ISO20022::CodeSet](#)

[ISO20022::DataTypes::Time](#)

[ISO20022::DataTypes::Year](#)

[ISO20022::DataTypes::Decimal](#)

[ISO20022::DataTypes::Rate](#)

[ISO20022::DataTypes::Quantity](#)

[ISO20022::DataTypes::Amount](#)

[ISO20022::DataTypes::Boolean](#)

[ISO20022::DataTypes::Indicator](#)

[ISO20022::DataTypes::Month](#)

[ISO20022::DataTypes::Duration](#)

[ISO20022::DataTypes::DateTime](#)

[ISO20022::DataTypes::MonthDay](#)

[ISO20022::DataTypes::Day](#)

[ISO20022::DataTypes::Date](#)

[ISO20022::LogicalLevel::MessageComponentType](#)

[ISO20022::LogicalLevel::MessageComponent](#)

[BIAN::Level2::BIANMessageComponent](#)

[ISO20022::LogicalLevel::ExternalSchema](#)

[ISO20022::LogicalLevel::ChoiceComponent](#)

[BIAN::Level2::BIANChoiceComponent](#)

[ISO20022::ConceptualLevel::Static::BusinessComponent](#)

[BIAN::Level1::BusinessObject](#)

[ISO20022::ConceptualLevel::Static::BusinessAssociation](#)

[ISO20022::ScopeLevel::BusinessRole](#)

[BIAN::Level1::ServiceRole](#)

[ISO20022::LogicalLevel::MessageElement](#)

[ISO20022::LogicalLevel::MessageAssociationEnd](#)

[BIAN::Level2::BIANMessageAssociationEnd](#)

[ISO20022::LogicalLevel::MessageAttribute](#)

[BIAN::Level2::BIANMessageAttribute](#)

[ISO20022::LogicalLevel::MessageBuildingBlock](#)

[BIAN::Level2::BIANMessageBuildingBlock](#)

[ISO20022::LogicalLevel::MessageDefinition](#)

[BIAN::Level2::BIANMessageDefinition](#)

[ISO20022::Constraint](#)

[BIAN::Level1::BIANConstraint](#)

[BIAN::Level2::Precondition](#)

[BIAN::Level2::Postcondition](#)

[BIAN::Level2::ServiceOperation](#)

[BIAN::Level2::ServiceGroup](#)

[BIAN::Level1::BIANBusinessArea](#)

[BIAN::Level1::Responsibility](#)

[BIAN::Level1::ResponsibilityFulfillment](#)

[BIAN::Level1::Capability](#)

[BIAN::Level1::RequestSatisfaction](#)

[BIAN::Level1::Notification](#)

[BIAN::Level1::Delegation](#)

[BIAN::Level1::BusinessDomain](#)

[BIAN::Level1::ServiceDomain](#)

[SemanticMetadata::SemanticRole](#)

[SemanticMetadata::BusinessContextType](#)

[SemanticMetadata::SemanticGrammarRole](#)

[SemanticMetadata::SemanticMetadataTag](#)

[SemanticMetadata::EmbeddedTag](#)

[SemanticMetadata::StandaloneTag](#)

[ISO20022::TopLevelCatalogueEntry](#)

[ISO20022::ConceptualLevel::Dynamic::BusinessTransaction](#)

[BIAN::Level1::BusinessScenario](#)

[ISO20022::ConceptualLevel::Dynamic::MessageTransportMode](#)

[ISO20022::ScopeLevel::BusinessProcess](#)

[ISO20022::LogicalLevel::BusinessArea](#)

[ISO20022::LogicalLevel::MessageChoreography](#)

[ISO20022::LogicalLevel::Reversing::ConvergenceDocumentation](#)

[ISO20022::LogicalLevel::Reversing::IndustryMessageSet](#)

[ISO20022::LogicalLevel::Reversing::ISO15022MessageSet](#)

[ISO20022::LogicalLevel::MessageSet](#)

[ISO20022::PhysicalLevel::SyntaxMessageScheme](#)

[ISO20022::Code](#)

[ISO20022::ConceptualLevel::Dynamic::Send](#)

[SemanticMetadata::TaggableElement](#)

[BIAN::Level2::Precondition](#)

[BIAN::Level2::BIANMessageDefinition](#)

[BIAN::Level2::Postcondition](#)

[BIAN::Level2::ServiceOperation](#)

[BIAN::Level2::ServiceGroup](#)

[BIAN::Level2::BIANMessageComponent](#)

[BIAN::Level2::BIANMessageBuildingBlock](#)

[BIAN::Level2::BIANMessageAttribute](#)

[BIAN::Level2::BIANMessageAssociationEnd](#)

[BIAN::Level2::BIANChoiceComponent](#)

[BIAN::Level3::ImplementationInterface](#)

[BIAN::Level3::ImplementationComponent](#)

[BIAN::Level1::BIANBusinessArea](#)

[BIAN::Level1::BusinessDomain](#)

[BIAN::Level1::ServiceDomain](#)

[BIAN::Level1::BusinessScenario](#)

[BIAN::Level1::BusinessObject](#)

[BIAN::Level1::BIANBusinessAttribute](#)

[BIAN::Level1::BIANBusinessAssociationEnd](#)

## Enumeration Hierarchy

[BIAN::Level2::InvocationKind](#)

[BIAN::Level1::MessageTransmissionKind](#)